SGOU
Higher Education for All

Quality Control

COURSE CODE: B21CA15DE

Bachelor of Computer Applications
Discipline Specific Elective Course
Self Learning Material



SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Quality Control

Course Code: B21CA15DE

Semester - VI

SGOU

Discipline Specific Elective Course
Undergraduate Programme
Bachelor of Computer Applications
Self Learning Material

(With Model Question Paper Sets)

വിദ്യകൊണ്ട് സ്വതന്ത്രരാവുക

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

QUALITY CONTROL

Course Code: B21CA15DE
Semester- VI
Discipline Specific Elective Course
Bachelor of Computer Applications

Academic Committee

Dr. M.V. Judy
Dr. Aji S.
Dr. Vishnukumar S.
Rajesh R.
Dr. Rafidha Rehiman K.A.
P.M. Ameera Mol
Dr. Ajitha R.S.
Dr. Bindu Lal T.S.
Dr. Sreeja S.

Development of the Content

Block 1: Subi Priya Laxmi S.B.N.
Block 2: Dr. Kanitha D.K.
Block 3: Greeshma P.P.
Block 4: Shamin S.
Block 5: Aswathy V.S.
Block 6: Sreerekha V.K.

Review and Edit

Dr. Shani Raj

Proofreading

Dr. Aravind S.G.

Scrutiny

Subi Priya Laxmi S.B.N.
Dr. Kanitha D.K.
Greeshma P.P.
Shamin S.
Aswathy V.S.
Sreerekha V. K.

Cover Design

Jobin J.

Co-ordination

Director (IC), MDDC :
Dr. Sajeevkumar G.
Coordinator, Development:
Dr. Anfal M.
Coordinator, Distribution:
Dr. Abdul Razak Kunnathodi

Scan this QR Code for reading the SLM on a digital device.

Edition
October 2026

Copyright
© Sreenarayanaguru Open University

ISBN 978-81-903304-8-0

9 788190 330480

SGOU

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from Sreenarayanaguru Open University. Printed and published on behalf of Sreenarayanaguru Open University by Registrar, SGOU, Kollam.

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear

With immense joy and excitement, I extend my heartfelt greetings to all of you and warmly welcome you to Sreenarayanaguru Open University.

Established in September 2020 as a state-driven initiative, Sreenarayanaguru Open University is dedicated to advancing higher education through open and distance learning. Our vision is guided by the principle of "access and quality define equity," laying the foundation for a celebration of excellence in education. I am delighted to share that we are steadfast in our commitment to uphold the highest standards and refrain from compromising on the quality of education we offer. The university draws its inspiration from the legacy of Sreenarayana Guru, a revered figure in the Indian renaissance movement. His name serves as a constant reminder for us to prioritize quality in all our academic endeavors.

Sreenarayanaguru Open University operates within the practical framework of the widely recognized "blended format." Acknowledging the constraints faced by distance learners in accessing traditional classroom settings, we have curated a pedagogical approach centered on three main components: Self Learning Material, Classroom Counselling, and Virtual Modes. This comprehensive blend is poised to deliver dynamic learning and teaching experiences, maximizing engagement and effectiveness. Our unwavering commitment to quality ensures excellence across all aspects of our educational initiatives.

The University aims to offer you an engaging and stimulating educational environment that fosters active learning. The SLM is designed to offer a comprehensive and cohesive learning experience, fostering a deep interest in the study of technological advancements in IT. Careful consideration has been given to ensure a logical progression of topics, facilitating a clear understanding of the discipline's evolution. The curriculum is thoughtfully crafted to provide ample opportunities for students to navigate through the current trends in information technology. Furthermore, this course is designed to provide essential insights into computer hardware, software classification, and foundational HTML concepts crucial for web development.

We assure you that the university student support services will closely stay with you for the redressal of your grievances during your studentship. Feel free to write to us about anything that seems relevant regarding the academic programme.

Wish you the best.



Regards,
Dr. Jagathy Raj V.P.

01-10-2026

SGOU

Contents

BLOCK 1 Software Quality Fundamentals and Quality Culture **1**
UNIT 1 Software Test Life Cycle 2
UNIT 2 Scope and Applications of Digital Marketing 19
UNIT 3 Business Models and Dimensions of Software Projects 31
UNIT 4 Software Ethics and Quality Management 51

BLOCK 2 Quality Management Standards and Models **73**
UNIT 1 Software Quality Standards and Business Models 74
UNIT 2 Review Techniques and Quality Frameworks 88
UNIT 3 Formal Review Methods 104
UNIT 4 Project Reviews and Agile Practices 117

BLOCK 3 Software Audits **130**
UNIT 1 Types of Audits 131
UNIT 2 Audit Process and the ISO 9001 Standard 151
UNIT 3 Audit According to the CMMI 167
UNIT 4 Software Quality Auditing 179

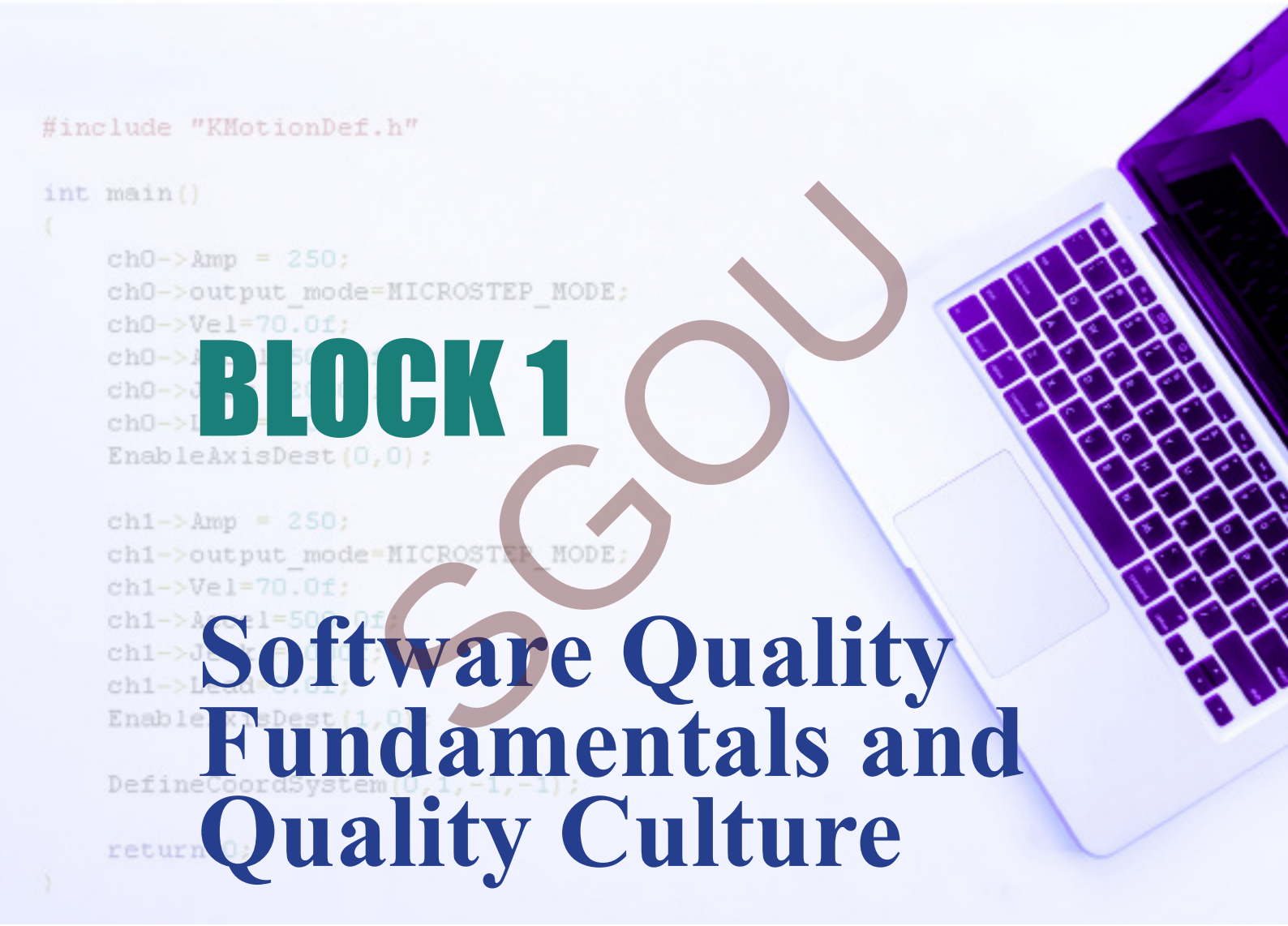
BLOCK 4 Verification and Validation **193**
UNIT 1 V&V Standards and Models 194
UNIT 2 IEEE 1012 and Integrity 220
UNIT 3 V&V for Requirements 233
UNIT 4 Software Standards and Validation 247

BLOCK 5 Measurement and Risk Management **269**
UNIT 1 Foundations of Software Measurement 270
UNIT 2 Software Measurement Models and Methods 295
UNIT 3 Measurement Tools and Risk Management Standards 334
UNIT 4 Practical Considerations for Risk Management 343

BLOCK 6 Testing Tools **360**
UNIT 1 Categories of Testing Tools 361
UNIT 2 Automated Testing Tools 380
UNIT 3 Mobile and API Testing 401
UNIT 4 Security and Validation Testing 425

Model Question Paper Sets **445**

SGOU



BLOCK 1

Software Quality Fundamentals and Quality Culture

SGOU



Software Test Life Cycle

Learning Outcomes

After completion of this unit, learner will be able to:

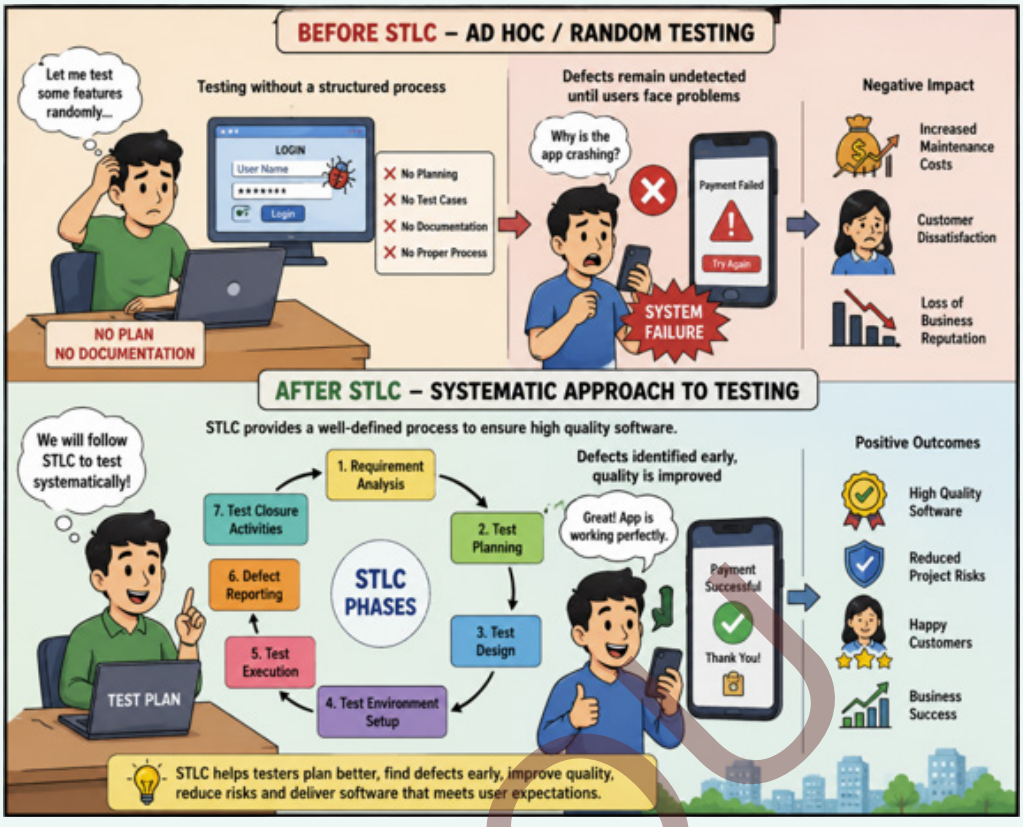
- explain the concept, objectives, and importance of the Software Test Life Cycle (STLC)
- describe the different phases of STLC and the activities performed in each phase
- differentiate between verification and validation, and explain test planning, test design, test execution, and defect reporting
- understand test closure activities and apply STLC concepts to real-world software testing scenarios

Prerequisite

Software has become an essential part of everyday life, supporting activities such as online banking, shopping, education, healthcare, and communication. As software applications become more complex, ensuring their correctness and reliability before release has become a major challenge. In the early days of software development, testing was often performed without a structured process. Testers relied on random or ad hoc testing, where features were checked without proper planning, documentation, or defined procedures. As a result, many defects remained undetected until the software was delivered to users, leading to system failures, increased maintenance costs, customer dissatisfaction, and loss of business reputation.

To overcome these limitations, the Software Test Life Cycle (STLC) was introduced as a systematic approach to software testing. STLC defines a sequence of well-planned testing activities, beginning with requirement analysis and ending with test closure. It helps testers design effective test cases, identify defects at an early stage, improve software quality, reduce project risks, and ensure that the software meets customer requirements. Understanding STLC is therefore essential for developing reliable, high-quality software and for delivering products that satisfy user expectations.

SGOU



Keywords

Software Test Life Cycle (STLC), Requirement Analysis, Test Planning, Test Design, Test Execution, Defect Reporting, Requirement Traceability Matrix (RTM), Test Closure

Discussion

1.1.1 Introduction to Software Test Life Cycle

Imagine that a company is developing an online food delivery application called FoodExpress. The application allows customers to browse restaurants, place food orders, make online payments, and track deliveries. The development team works hard to build these features, but an important question remains: *How can we ensure that the application works correctly before it is released to customers?*

Suppose a customer tries to place an order and the application crashes during payment. Another customer may be unable to log in even after entering the correct password. Such problems can lead to customer dissatisfaction, financial losses, and damage to the company's reputation. Therefore, software must be thoroughly tested before it is delivered to users.

SGOU

Software testing is not a random activity. It follows a systematic process known as the Software Test Life Cycle (STLC). STLC provides a structured framework that guides testing activities from understanding requirements to formally closing the testing process. Each phase of STLC has specific objectives, inputs, outputs, and deliverables that help ensure the quality of the software product.

The Software Test Life Cycle enables testing teams to identify defects early, reduce risks, improve software reliability, and ensure that the final product satisfies user expectations. By following a well-defined testing process, organisations can deliver high-quality software in a cost effective and efficient manner.

1.1.1.1 What is the Software Test Life Cycle?

The Software Test Life Cycle (STLC) is a sequence of activities performed during software testing to ensure that the software meets specified requirements and quality standards. It defines a systematic approach for planning, designing, executing, and evaluating testing activities.

STLC begins when software requirements become available and continues until testing activities are completed and formally closed. Each phase has clearly defined goals and deliverables that help maintain consistency throughout the testing process.

In simple terms, STLC can be viewed as a roadmap for software testing. Just as a traveller follows a planned route to reach a destination, testers follow the stages of STLC to verify that software functions correctly and satisfies user needs.

The major phases of STLC are:

1. Requirement Analysis
2. Test Planning
3. Test Design
4. Test Environment Setup
5. Test Execution
6. Defect Reporting
7. Test Closure Activities

These phases work together to ensure comprehensive testing coverage and effective defect detection. Each phase contributes to the overall quality of the software and helps the testing team achieve its objectives in an organised manner.

1.1.1.2 Objectives of STLC

The primary objective of STLC is to ensure that software functions correctly and meets user expectations. However, STLC serves several other important purposes within the software development process.

SGOU

One objective is to identify defects as early as possible. Detecting defects during testing is significantly less expensive than discovering them after the software has been released. Early defect detection reduces maintenance costs and improves software quality.

Another objective is to verify that all requirements have been implemented correctly. Testing ensures that every feature described in the Software Requirements Specification (SRS) document behaves as expected.

STLC also aims to improve software reliability. By testing different scenarios, inputs, and conditions, testers can identify weaknesses that might affect system performance or stability.

An additional objective is to reduce project risk. Systematic testing helps uncover issues related to functionality, security, performance, and usability before they impact real users.

The major objectives of STLC include:

- Ensuring software quality.
- Detecting defects and failures.
- Verifying functional and non-functional requirements.
- Improving customer satisfaction.
- Reducing project risks.
- Supporting successful software release.
- Increasing confidence in the software product.

For example, in the FoodExpress application, testing verifies that users can register, search for restaurants, place orders, complete payments, and track deliveries without encountering errors.

1.1.1.3 Importance of STLC

Modern software systems are becoming increasingly complex. Applications often support thousands or even millions of users and involve numerous interconnected components. Without a structured testing process, ensuring software quality becomes extremely difficult.

STLC provides a disciplined approach that helps organisations manage testing activities effectively. Each phase focuses on specific tasks and deliverables, reducing confusion and improving coordination among team members.

One important benefit of STLC is improved defect management. Since testing activities are organised systematically, defects can be identified, documented, tracked, and resolved more efficiently.

STLC also improves communication between testers, developers, business analysts, and project managers. Clearly defined phases and deliverables ensure that everyone understands their responsibilities and expectations.

SGOU

Another significant advantage is cost reduction. Defects discovered early in the software life cycle are generally less expensive to fix than defects discovered after deployment.

Consider the FoodExpress application again. Suppose testers identify a payment processing defect during testing. The issue can be corrected before customers begin using the application. If the same defect were discovered after deployment, it could result in failed transactions, customer complaints, and financial losses.

Thus, STLC plays a vital role in delivering reliable, secure, and user-friendly software products.

1.1.1.4 STLC vs SDLC

Students often confuse the Software Development Life Cycle (SDLC) with the Software Test Life Cycle (STLC). Although both processes are closely related, they serve different purposes.

The Software Development Life Cycle focuses on developing the software product, while the Software Test Life Cycle focuses on testing and validating that product.

Table 1.1.1 SDLC VS STLC

Aspect	SDLC	STLC
Full Form	Software Development Life Cycle	Software Test Life Cycle
Purpose	Develop software	Test software
Focus	Building the product	Validating the product
Participants	Developers, Analysts, Designers	Testers, QA Engineers
Output	Software Application	Tested and Verified Software
Goal	Product Creation	Quality Assurance

The two life cycles work together throughout a project. As developers build software during SDLC, testers perform testing activities through STLC to ensure that the software meets quality standards. While SDLC focuses on creating the product, STLC focuses on ensuring that the product functions correctly and satisfies user requirements.

1.1.1.5 Verification and Validation

Two important concepts in software testing are Verification and Validation. Although they are related, they represent different activities.

Verification answers the question:

"Are we building the product right?"

Verification involves evaluating documents, designs, requirements, and code without executing the software. The objective is to ensure that development activities follow established standards and specifications.

SGOU

Examples of verification activities include:

- Requirement reviews
- Design reviews
- Code inspections
- Walkthroughs
- Peer reviews

Suppose the FoodExpress team reviews the Software Requirements Specification document to confirm that all customer requirements have been documented correctly. This activity is an example of verification.

Validation answers the question:

"Are we building the right product?"

Validation involves executing the software to ensure that it satisfies user needs and performs expected functions.

Examples of validation activities include:

- Functional testing
- System testing
- User acceptance testing
- Performance testing

Suppose a tester executes the login feature of FoodExpress and verifies that valid users can successfully access their accounts. This activity is an example of validation.

Table 1.1.2 Verification vs Validation

Verification	Validation
Evaluates documents and designs	Evaluates running software
Does not execute software	Executes software
Detects errors early	Detects functional defects
Focuses on process	Focuses on product
"Are we building the product right?"	"Are we building the right product?"

Both verification and validation are essential for producing high-quality software. Verification helps prevent defects, while validation helps detect defects before release.

SGOU

1.1.2 Phases of Software Test Life Cycle (STLC)

The Software Test Life Cycle (STLC) consists of a series of well-defined phases that guide testing activities from understanding software requirements to formally closing the testing process. Each phase has a specific objective and produces deliverables that serve as inputs for the next phase. By following these phases systematically, software organisations can ensure that testing is planned, organised, and effective.

The phases of STLC are interconnected. The successful completion of one phase forms the foundation for the next. For example, testers cannot design effective test cases without first understanding the software requirements. Similarly, test execution cannot begin until the testing environment has been properly prepared.

In the FoodExpress application, the testing team first studies the customer requirements, then plans the testing activities, designs test cases, prepares the testing environment, executes the tests, reports defects, and finally closes the testing process after ensuring that the software satisfies all quality standards.

1. Requirement Analysis

Requirement Analysis is the first phase of the Software Testing Life Cycle. In this phase, the testing team carefully studies the Software Requirement Specification (SRS), business requirement documents, and other project documents to understand what the software is expected to do. The primary objective is to identify the requirements that can be tested and to determine the scope of testing. Testers also communicate with clients, business analysts, and developers to clarify any unclear or incomplete requirements. Based on this analysis, they identify the types of testing required and prepare the Requirement Traceability Matrix (RTM), which links software requirements with corresponding test cases.

Example: Before constructing a building, an architect carefully studies the building plan to understand the design requirements. Similarly, testers study the software requirements before planning the testing activities.

SGOU

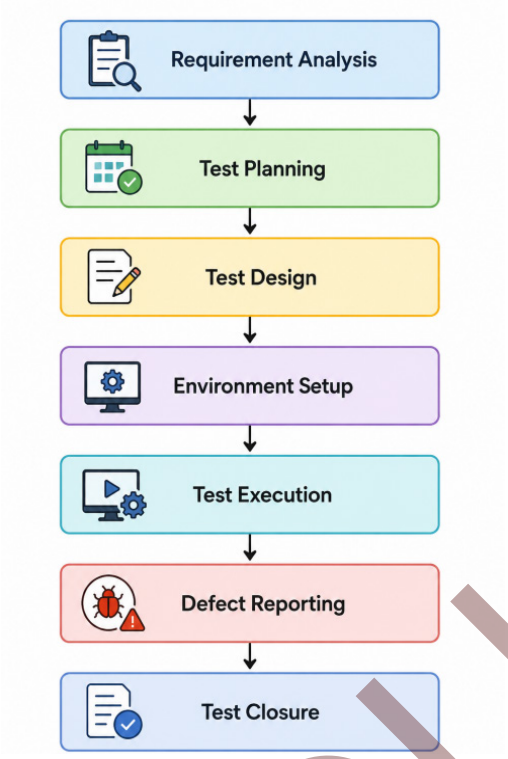


Fig 1.1.1 Seven phases of STLC

2. Test Planning

Test Planning is the second phase of STLC and is considered one of the most important phases because it establishes the overall testing strategy. During this phase, the test manager prepares a detailed Test Plan that defines the testing objectives, testing approach, required resources, testing tools, schedule, budget, and responsibilities of team members. Risks associated with testing are also identified, and suitable mitigation plans are prepared. A well-prepared test plan ensures that testing activities are organised, efficient, and completed within the available time and budget.

Example: Before organising a college cultural festival, the organising committee prepares a detailed plan that includes the budget, schedule, venue, and responsibilities. Similarly, software testers prepare a test plan before beginning the testing process.

3. Test Case Design

During the Test Case Design phase, the testing team prepares detailed test cases based on the software requirements and test scenarios. A test case defines the test steps, input data, expected output, and test conditions required to verify a particular functionality of the software. Testers design both positive and negative test cases to check how the software behaves under normal and unexpected conditions. The test cases are reviewed to ensure that all functional and non-functional requirements are adequately covered. This phase provides a systematic approach for verifying whether the software behaves as expected.

Example: Before conducting an examination, a teacher prepares questions that cover all the important topics in the syllabus. Similarly, testers design test cases to cover all the requirements and functionalities of the software.

SGOU

4. Test Environment Setup

The **Test Environment Setup** phase involves preparing the environment in which the software will be tested. This includes configuring hardware, installing software, setting up databases, networks, servers, and other required resources. The testing environment should closely resemble the actual environment in which the software will be used. Before testing begins, the environment is verified to ensure that all components are functioning correctly. A properly configured test environment helps identify defects that may occur during real-world usage.

Example: Before conducting a laboratory experiment, all laboratory equipment and materials are arranged and tested to ensure they work correctly. Similarly, the testing environment is prepared before executing software tests.

5. Test Execution

Test Execution is the phase in which testers execute the prepared test cases on the software. During execution, the actual output produced by the software is compared with the expected output specified in the test cases. If any differences are found, they are recorded as defects or bugs. Testers also perform retesting after defects are corrected and conduct regression testing to ensure that recent changes have not introduced new defects. This phase verifies whether the software performs according to the specified requirements.

Example: A teacher evaluates students' answers during an examination to determine whether they have answered correctly. Similarly, testers execute test cases to verify whether the software functions correctly.

6. Defect Reporting and Tracking

Whenever defects are identified during test execution, they are documented in the Defect Reporting and Tracking phase. Each defect is recorded with details such as its description, severity, priority, steps to reproduce the problem, and expected results. The defects are then assigned to developers for correction. After the developers fix the defects, testers verify the corrections and update the defect status until the issue is completely resolved. This process ensures that every identified defect is properly monitored and addressed before software release.

Example: If a manufacturing company identifies defective products during inspection, the defects are recorded, corrected, and rechecked before the products reach customers. Software defect tracking follows a similar process.

7. Test Closure

Test Closure is the final phase of the Software Testing Life Cycle. Once all planned testing activities have been completed, the testing team evaluates whether the testing objectives have been achieved. A Test Summary Report is prepared, describing the testing activities performed, defects identified and resolved, test coverage, and overall software quality. The team also documents lessons learned, archives testing documents for future reference, and obtains formal approval or sign-off from stakeholders. This phase marks the official completion of the testing process.

SGOU

Example: After a university examination is completed, the results are published, reports are prepared, and examination records are archived. Similarly, software testing concludes with documentation, reporting, and formal project closure.

1.1.3 Test Planning and Design

Test Planning and Test Design are two important activities in STLC that ensure testing is systematic and effective. Although they are consecutive phases, they serve different purposes.

Test Planning focuses on deciding how testing will be performed. It includes defining the scope of testing, selecting testing techniques, estimating effort, allocating resources, scheduling activities, identifying risks, and preparing the Test Plan.

Test Design, on the other hand, focuses on what will be tested. During this phase, testers create test scenarios, write detailed test cases, prepare test data, and ensure that every requirement is covered using the Requirement Traceability Matrix (RTM).

Together, Test Planning and Test Design provide a strong foundation for successful software testing.

Table 1.1.3 Difference Between Test Planning and Test Design

Test Planning	Test Design
Defines the testing strategy	Creates test cases and test scenarios
Performed by Test Manager	Performed by Test Engineers
Produces Test Plan	Produces Test Cases and Test Data
Focuses on process	Focuses on implementation

1.1.4 Test Execution and Defect Reporting

Test Execution and Defect Reporting work together to evaluate software quality. During Test Execution, testers execute the prepared test cases and compare the actual results with the expected results. Whenever a discrepancy is found, it is recorded as a defect.

Each reported defect is assigned a severity and priority level, tracked using a defect management tool, corrected by developers, and verified through retesting. Regression testing is also performed to ensure that software modifications have not introduced new defects.

This phase provides valuable information about software quality and helps developers improve the product before release.

SGOU



Fig 1.1.2 Software Testing Closure Process

1.1.4.1 Test Execution

Test Execution is the phase where actual testing is performed. In this phase, testers execute the test cases prepared earlier and compare the actual output with the expected output. The goal is to verify whether the application behaves as per requirements.

Each test case is marked as Pass or Fail based on the result obtained during execution.

Why is Test Execution Important?

This is the most critical phase of STLC because it directly identifies defects in the system. It helps determine whether the software is ready for release or needs further fixes. Test Execution ensures that all functionalities are validated in real conditions.

Activities Performed

- Executing test cases step by step.
- Recording actual results.
- Comparing actual and expected outcomes.
- Reporting failed test cases.
- Retesting after fixes are applied.

SGOU

In the FoodExpress application, a tester executes the login test case. The expected result is that the user should successfully reach the dashboard after entering valid credentials. However, instead of navigating to the dashboard, the application crashes. This indicates a failure, and the test case is marked as failed.

Similar execution is done for ordering food and making payments to ensure that all functionalities are working properly.

1.1.4.2 Defect Reporting

Defect Reporting is the process of documenting and communicating issues found during test execution. A defect is any mismatch between expected and actual behaviour of the application. This phase ensures that developers clearly understand the problem and can fix it efficiently.

Why is Defect Reporting Important?

Without proper defect reporting, developers may not understand the issue clearly, leading to delays or incorrect fixes. A well written defect report improves communication between testers and developers and ensures faster resolution of issues.

Activities Performed

- Identifying failed test cases.
- Logging defects in defect tracking tools.
- Assigning severity and priority levels.
- Providing reproduction steps.
- Attaching screenshots or logs.
- Tracking defect lifecycle until closure.

Severity: How badly the defect impacts the *application's functionality* (e.g., an application crash is a High Severity issue).

Priority: How fast the business/developers need to *fix the defect* (e.g., A typo on the main landing page logo has Low Severity but High Priority because it affects brand image).

While testing FoodExpress, a tester finds that the application crashes when the user clicks the "Pay Now" button. The tester records this issue in a defect tracking tool with all necessary details such as steps to reproduce, expected result, actual result, and severity level. Since payment is a critical function, the defect is marked as high severity and high priority.

The developer fixes the issue, and the tester verifies it again through retesting.

SGOU

1.1.5 Test Closure Activities

Test Closure is the final phase of STLC where testing activities are formally completed. In this phase, the team evaluates the overall testing process and confirms whether the software is ready for release.

Why is Test Closure Important?

Test Closure helps measure the effectiveness of testing activities. It ensures that all planned testing has been completed and provides insights for future improvements. It also ensures proper documentation and project closure.

Activities Performed

- Verifying test completion criteria.
- Preparing test summary reports.
- Analysing test results and defect metrics.
- Documenting lessons learned.
- Archiving test cases and reports.
- Obtaining final approval for release.

After completing testing of FoodExpress, the team reviews the results. They find that 500 test cases were executed, 60 defects were identified, and all critical defects were fixed successfully. A final test summary report is prepared, highlighting testing results and quality metrics.

The team also discusses what improvements can be made in future projects, such as improving automation coverage and reducing defect leakage. Finally, all documents are stored safely, and the application is approved for release.

SGOU

Recap

- Software testing ensures that software meets specified requirements and quality standards.
- STLC is a structured process consisting of seven phases, from Requirement Analysis to Test Closure.
- Requirement Analysis helps testers understand and verify software requirements.
- Test Planning defines the testing strategy, schedule, resources, and scope.
- Test Design involves creating test scenarios, test cases, and test data.

- Test Environment Setup prepares the hardware and software required for testing.
- Test Execution verifies software functionality by executing test cases.
- Defect Reporting records and tracks software defects for correction.
- Test Closure evaluates testing activities, prepares reports, and formally completes the testing process.
- Verification checks whether the software is built correctly, while Validation checks whether the correct software is built.

Objective Type Questions

1. What is the full form of STLC?
2. Which document contains software requirements?
3. Which STLC phase comes first?
4. Who prepares the Test Plan?
5. What is the main output of the Test Planning phase?
6. Which document links requirements with test cases?
7. What does RTM stand for?
8. Which phase involves creating test cases?
9. Which phase prepares the testing environment?
10. Which testing checks whether the software build is stable?
11. What is the process of executing test cases called?
12. What is a mismatch between expected and actual results called?
13. Which testing verifies that a fixed defect has been corrected?
14. Which testing ensures that existing functionality is not affected after changes?
15. What is the final phase of STLC?
16. Which report summarises the overall testing activities?

SGOU

17. Verification answers which question?

18. Validation answers which question?

19. Which life cycle focuses on software development?

20. Which life cycle focuses on software testing?

Answers to Objective Type Questions

1. Software Test Life Cycle.
2. Software Requirements Specification (SRS).
3. Requirement Analysis.
4. Test Manager / Test Lead.
5. Test Plan Document.
6. Requirement Traceability Matrix (RTM).
7. Requirement Traceability Matrix.
8. Test Design.
9. Test Environment Setup.
10. Smoke Testing.
11. Test Execution.
12. Defect (Bug).
13. Retesting.
14. Regression Testing.
15. Test Closure.
16. Test Closure Report (Test Summary Report).
17. Are we building the product right?
18. Are we building the right product?
19. Software Development Life Cycle (SDLC).
20. Software Test Life Cycle (STLC).

SGOU

Assignments

1. Explain the Software Test Life Cycle (STLC) and discuss each phase with suitable examples.

2. Describe the Requirement Analysis phase of STLC. Explain its importance, activities, inputs, outputs, and deliverables.

3. Explain Test Planning in detail. Discuss the components of a Test Plan and its importance in software testing.

4. Describe the Test Design phase. Explain Test Scenarios, Test Cases, Test Data, and the Requirement Traceability Matrix (RTM).

5. Explain Test Execution and Defect Reporting. Differentiate between Retesting and Regression Testing with suitable examples.

6. Discuss Test Closure Activities in detail. Explain Test Summary Report, Test Metrics, and Project Sign-off.

7. Differentiate between Software Development Life Cycle (SDLC) and Software Test Life Cycle (STLC). Also explain Verification and Validation with examples.

Suggested Reading

1. Kaner, C., Falk, J., & Nguyen, H. Q.. (1999). *Testing computer software* (2nd ed.). Wiley.

2. Patton, R.. (2006). *Software testing* (2nd ed.). Sams Publishing.

3. Lewis, W. E.. (2016). *Software testing and continuous quality improvement* (4th ed.). CRC Press.

4. Jorgensen, P. C.. (2013). *Software testing: A craftsman's approach* (4th ed.). CRC Press.

SGOU

Reference

1. Pressman, R. S., & Maxim, B. R.. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

2. Sommerville, I.. (2016). *Software engineering* (10th ed.). Pearson.

3. Myers, G. J., Sandler, C., & Badgett, T.. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.

4. Ammann, P., & Offutt, J.. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.

SGOU



Scope and Applications of Digital Marketing

Learning Outcomes

After completion of this unit, learner will be able to:

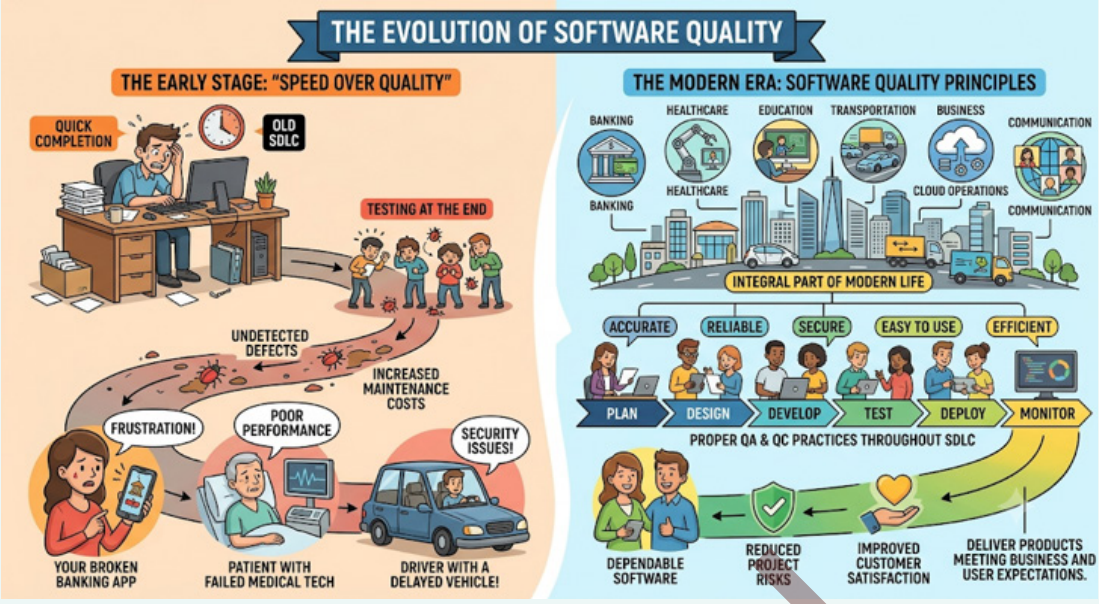
- define software quality and explain its importance in software development
- describe the major quality attributes of software with suitable examples
- differentiate between Quality Assurance (QA) and Quality Control (QC)
- explain how software quality contributes to customer satisfaction and business success

Prerequisite

Software has become an integral part of modern life and is widely used in banking, healthcare, education, transportation, business, and communication. As software systems become larger and more complex, users expect them to be accurate, reliable, secure, and easy to use. In the early stages of software development, organisations mainly focused on completing software quickly, with little attention given to quality. Testing was often performed only at the end of development, resulting in undetected defects, increased maintenance costs, poor performance, security issues, and customer dissatisfaction.

To overcome these limitations, the concept of Software Quality was introduced. Software quality focuses on developing software that not only meets functional requirements but also performs efficiently, reliably, securely, and consistently. It also emphasises following proper quality assurance and quality control practices throughout the software development process. Studying software quality helps developers and testers build dependable software, reduce project risks, improve customer satisfaction, and deliver products that meet both business and user expectations.

SGOU



Keywords

Software Quality, Reliability, Usability, Efficiency, Security, Maintainability, Quality Assurance (QA), Quality Control (QC)

Discussion

1.2.1 Introduction

Imagine that you have downloaded a mobile banking application. The application opens quickly, allows you to transfer money securely, and never crashes while performing transactions. On the other hand, consider an application that frequently freezes, displays incorrect account balances, and takes a long time to load. Which application would you trust?

Most users would choose the first application because it demonstrates high software quality. In today's digital world, software is used in banking, healthcare, education, transportation, and entertainment. Therefore, ensuring software quality is essential for user satisfaction and business success.

1.2.2 Definition of Software Quality

Software Quality refers to the degree to which a software product satisfies specified requirements and meets user expectations while performing its intended functions correctly, efficiently, and reliably.

SGOU

According to the IEEE, software quality is:

"The degree to which a system, component, or process meets specified requirements and customer needs or expectations."

In simple terms, software quality means creating software that works correctly, is easy to use, performs efficiently, and provides value to users.

Understanding Software Quality

Consider purchasing a new car. A quality car should:

- Start immediately when needed.
- Consume fuel efficiently.
- Provide safety features.
- Be comfortable to drive.
- Require minimal repairs.

Similarly, quality software should:

- Function correctly.
- Operate efficiently.
- Protect user data.
- Be easy to use.
- Require fewer corrections after release.

Just as customers evaluate the quality of a car based on performance and reliability, users evaluate software quality based on their experience while using it.

1.2.2.1 Characteristics of Good Software Quality

A quality software product should:

- Meet user requirements.
- Produce accurate results.
- Be reliable and dependable.
- Perform tasks efficiently.
- Protect data and resources.

SGOU

- Be easy to maintain and update.
- Provide a user-friendly interface.

Software quality is important because:

1. Improves Customer Satisfaction

Users prefer software that performs accurately and consistently.

Example: Online shopping applications such as Amazon must display correct product information and process orders successfully.

2. Reduces Maintenance Cost

Quality software contains fewer defects, reducing the effort required to fix problems.

Example: Detecting an error during development is cheaper than fixing it after the software has been deployed to thousands of users.

3. Enhances Reliability

Reliable software continues to function correctly under different conditions.

Example: ATM software must operate correctly every time a customer performs a transaction.

4. Protects Business Reputation

Poor-quality software can damage an organisation's image.

Example: A banking application that frequently crashes may cause customers to lose trust in the bank.

5. Ensures Safety

In critical systems, software quality can directly affect human lives.

Example: Software used in hospitals for monitoring patients must provide accurate information at all times.

Software quality is the ability of a software product to meet requirements and satisfy user expectations. High-quality software is reliable, efficient, secure, and easy to use. Maintaining software quality helps organisations reduce costs, improve customer satisfaction, and ensure dependable operation.

1.2.3 Quality Attributes

Suppose two students submit assignments using different word-processing applications. Both applications allow users to create documents, but one application is fast, easy to use, and never crashes, while the other is slow and confusing.

Although both applications perform the same basic function, the first application provides a better user experience because of its superior quality attributes. Quality

SGOU

attributes help us measure and evaluate different aspects of software quality.

Quality Attributes are the characteristics or properties of software that determine how well the software performs its intended functions.

They describe the non-functional aspects of software, such as reliability, usability, efficiency, and security.

In simple words, quality attributes answer the question:

"How well does the software perform its job?"

1.2.3.1 Major Quality Attributes

1. Reliability

Reliability refers to the ability of software to operate correctly without failure for a specified period.

Consider an ATM, customers expect the ATM to function correctly every time they withdraw money. If the ATM frequently crashes, users will lose confidence in the system. Similarly, reliable software consistently produces correct results.

2. Usability

Usability refers to how easy the software is to learn, understand, and use.

Think about using a food delivery application; users should be able to search restaurants, place orders, and make payments without confusion. If users require extensive training to use the application, usability is poor. A user-friendly interface increases software acceptance.

3. Efficiency

Efficiency refers to how effectively software uses system resources such as memory, processing power, and network bandwidth.

A video streaming application should load videos quickly and use internet bandwidth efficiently. Software that consumes excessive resources may become slow and unresponsive.

4. Security

Security refers to protecting software and data from unauthorised access, modification, or destruction.

Online banking systems store sensitive financial information; strong passwords, encryption, and authentication mechanisms help protect customer data from cyber threats.

5. Maintainability

Maintainability refers to the ease with which software can be modified, corrected, or enhanced.

SGOU

A college management system may require updates every academic year. If developers can easily add new features and fix issues, the software has good maintainability.

6. Portability

Portability refers to the ability of software to operate on different hardware or operating systems.

A web browser that works on Windows, Linux, Android, and iOS demonstrates high portability. Users can access the software regardless of the device they use.

7. Scalability

Scalability refers to the ability of software to handle increasing workloads without significant performance degradation.

During online ticket booking for a popular event, thousands of users may access the system simultaneously. A scalable system continues to perform efficiently even under heavy traffic.

Relationship Between Quality Attributes

Quality attributes often work together to create a better software product.

For example:

- A banking application should be reliable and secure.
- An e-learning platform should be usable and efficient.
- An e-commerce website should be scalable and maintainable.

Developers must balance these attributes according to project requirements.

Quality attributes are measurable characteristics that determine the overall quality of software. Important quality attributes include reliability, usability, efficiency, security, maintainability, portability, and scalability. Together, these attributes ensure that software performs effectively and satisfies user expectations.

1.2.4 Quality Assurance (QA) vs Quality Control (QC)

Imagine a bakery that produces cakes every day. The bakery owner wants every cake to taste good and look attractive. To achieve this:

- The bakery follows standard recipes and baking procedures.
- Each cake is inspected before being sold.

The first activity represents Quality Assurance (QA), while the second represents Quality Control (QC). The same concept applies to software development.

SGOU

1.2.4.1 Quality Assurance (QA)

Quality Assurance (QA) is a systematic process that focuses on preventing defects by improving development processes and standards. QA ensures that the right methods, procedures, and practices are followed throughout software development. QA is process-oriented and focuses on defect prevention. Consider a university examination system. Before conducting examinations, the university:

- Defines examination rules.
- Prepares guidelines for question paper creation.
- Establishes evaluation procedures.
- Trains invigilators.

These activities help prevent problems before examinations begin. Similarly, software QA establishes standards and procedures to prevent defects.

Activities in Quality Assurance

QA activities include:

- Defining development standards.
- Conducting process reviews.
- Creating quality plans.
- Performing audits.
- Providing staff training.
- Improving development methodologies.

1.2.4.2 Quality Control (QC)

Quality Control (QC) is the process of identifying and correcting defects in the software product. QC focuses on examining the actual product to verify that it meets specified requirements. QC is product-oriented and focuses on defect detection. Returning to the bakery example: Before selling cakes, inspectors check:

- Shape and appearance.
- Taste and texture.
- Packaging quality.

Any defective cake is removed. Similarly, software testers execute test cases and identify defects before software is released.

SGOU

Activities in Quality Control

QC activities include:

- Software testing.
- Defect identification.
- Code inspection.
- Product reviews.
- Verification of requirements.
- Bug reporting and correction.

Table 1.2.1 Difference Between Quality Assurance and Quality Control

Quality Assurance (QA)	Quality Control (QC)
Focuses on processes.	Focuses on products.
Prevents defects.	Detects defects.
Proactive approach.	Reactive approach.
Conducted throughout development.	Conducted on completed work products.
Includes audits and process improvement.	Includes testing and inspection.
Ensures quality is built into software.	Ensures software meets quality requirements.

Consider constructing a house.

Quality Assurance:

- Creating construction guidelines.
- Selecting standard materials.
- Training workers.
- Following approved designs.

Quality Control:

- Inspecting walls after construction.
- Checking electrical wiring.
- Verifying plumbing connections.
- Identifying construction defects.

SGOU

Thus, QA helps prevent mistakes, while QC finds mistakes that already exist.

Why Both QA and QC are Necessary

Neither QA nor QC alone is sufficient.

- QA reduces the chances of defects occurring.
- QC identifies defects that still manage to appear.

Together they help deliver reliable, secure, and high-quality software.

For example, a banking application may follow strict development standards (QA), while testers perform extensive testing before release (QC). The combination ensures both process quality and product quality.

Quality Assurance (QA) focuses on preventing defects through proper processes, standards, and practices. Quality Control (QC) focuses on detecting and correcting defects in the software product. QA is process-oriented, while QC is product-oriented. Both are essential for delivering high-quality software that meets user expectations and business requirements.

Recap

- Software quality refers to the ability of software to satisfy specified requirements and user expectations.
- High-quality software is reliable, efficient, secure, and user-friendly.
- Quality attributes include reliability, usability, efficiency, security, maintainability, portability, and scalability.
- Reliability ensures software works correctly without failure.
- Usability makes software easy to learn and operate.
- Efficiency ensures optimum use of system resources.
- Security protects software and user data from unauthorised access.
- Maintainability enables easy modification and enhancement of software.
- Quality Assurance (QA) focuses on preventing defects by improving development processes.
- Quality Control (QC) focuses on detecting and correcting defects through testing and inspection.

SGOU

Objective Type Questions

1. What is software quality?
2. Which organisation defined software quality as meeting specified requirements and customer expectations?
3. Which quality attribute ensures software operates without failure?
4. Which quality attribute measures ease of use?
5. Which quality attribute protects software from unauthorised access?
6. Which quality attribute measures resource utilisation?
7. Which quality attribute allows software to run on different platforms?
8. Which quality attribute allows software to handle increased workload?
9. Which quality attribute makes software easy to modify?
10. What is the full form of QA?
11. What is the full form of QC?
12. Which process focuses on defect prevention?
13. Which process focuses on defect detection?
14. QA is process-oriented. (True/False)
15. QC is product-oriented. (True/False)

SGOU

Answers to Objective Type Questions

1. Degree to which software meets requirements and user expectations.
2. IEEE.
3. Reliability.
4. Usability.
5. Security.
6. Efficiency.
7. Portability.

8. Scalability.
9. Maintainability.
10. Quality Assurance.
11. Quality Control.
12. Quality Assurance.
13. Quality Control.
14. True.
15. True.

Assignments

1. Define software quality. Explain its characteristics and importance with suitable examples.
2. Explain the major quality attributes of software with real-life examples.
3. Discuss the importance of software quality in modern software development.
4. Differentiate between Quality Assurance (QA) and Quality Control (QC) with suitable examples.
5. Explain the activities involved in Quality Assurance and Quality Control.
6. Discuss the relationship between software quality attributes and software performance.
7. Explain how QA and QC together help in delivering high-quality software.

Suggested Reading

1. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.
2. Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.
3. Lewis, W. E. (2016). *Software testing and continuous quality improvement* (4th ed.). CRC Press.
4. Kan, S. H. (2002). *Metrics and models in software quality engineering* (2nd ed.). Addison-Wesley.

SGOU

Reference

1. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education.
2. Sommerville, I. (2016). Software engineering (10th ed.). Pearson.
3. International Organization for Standardization. (2011). ISO/IEC 25010:2011 systems and software engineering—Systems and software quality requirements and evaluation (SQuaRE)—System and software quality models.
4. IEEE Computer Society. (2017). IEEE Standard for System and Software Verification and Validation (IEEE Std 1012-2016). IEEE.

SGOU



Business Models and Dimensions of Software Projects

Learning Outcomes

After completion of this unit, learner will be able to:

- explain software business models, revenue models, and licensing strategies used in software organisations
- describe the five dimensions of a software project and their role in project management
- analyse the relationship between project dimensions and explain the need for balancing project constraints
- apply appropriate business and project management concepts for successful software development

Prerequisite

Software organisations today operate in highly competitive environments where they must deliver high quality products while also ensuring financial sustainability and successful project completion. In the early stages of the software industry, many organisations focused mainly on developing software without giving sufficient attention to business models, revenue generation, licensing policies, project planning, and balancing project constraints. As a result, many software projects failed due to poor financial planning, unclear project scope, unrealistic schedules, cost overruns, inefficient resource utilisation, and inappropriate licensing decisions. Even technically sound software products often failed because organisations could not sustain development or satisfy customer expectations.

To overcome these limitations, software engineering has evolved to include concepts such as software business models, revenue models, licensing strategies, project dimensions, and project constraint management. These concepts help organisations understand how software creates value, generates income, protects intellectual property, and delivers successful projects within available time, cost, quality, and resource limitations. Studying these topics enables learners to understand not only how software is developed but also how software businesses operate successfully. This knowledge helps future software professionals make informed business and project management decisions, improve customer satisfaction, reduce project risks, and contribute to the long-term success of software organisations.

SGOU

Keywords

Software Business Model, Revenue Model, Licensing Strategy, Scope, Time, Cost, Quality, Resources

Discussion

1.3.1 Introduction

Imagine that a group of software developers creates a mobile application that helps students prepare for examinations. The application becomes popular among learners because it provides study materials, practice tests, and progress tracking features. As the number of users grows, the developers need to maintain servers, fix bugs, provide customer support, and introduce new features. All these activities require money and resources. Therefore, the developers must decide how the software will generate income while continuing to provide value to users.

This decision is guided by a Software Business Model. A business model is not simply about selling software; it defines how a company creates value, delivers that value to customers, and earns revenue in return. The success of many software companies depends not only on the quality of their products but also on choosing an appropriate business model.

1.3.2 Software Business Model

A Software Business Model is the strategy used by an organisation to develop, market, deliver, and monetise software products or services. It describes the relationship between the software provider and its customers, explaining how the company generates revenue while satisfying customer needs.

In simple terms, a software business model answers three important questions:

1. What value does the software provide?
2. Who are the customers?
3. How will the company earn money from the software?

The answers to these questions help organisations build sustainable businesses and plan future growth.

Consider a transportation company. Some transportation services operate by selling bus tickets, some offer monthly travel passes, while others provide premium services for special customers. Although all these organisations help people travel from one place to another, they generate revenue in different ways.

Similarly, software companies may provide similar types of software but use different methods to earn income. One company may sell software directly, another may offer

SGOU

subscription based access, while a third may provide the software free of charge and earn revenue through advertisements. These different approaches represent different software business models.

1.3.2.1 Why Software Business Models Are Important

A software business model provides a roadmap for how a software company operates and survives in a competitive market. Without a clear business model, even technically excellent software may fail because the organisation cannot generate enough revenue to support development and maintenance activities.

A well-designed business model helps companies identify their target customers and understand how to meet customer expectations effectively. It also assists in planning investments, managing operational costs, and ensuring long term sustainability. For startups, selecting the right business model can be the difference between success and failure.

Furthermore, software business models influence product design decisions. For example, a company following a subscription model may focus on continuous updates and customer retention, whereas a company using a one time purchase model may emphasise delivering a complete product at the time of sale.

1.3.2.2 Types of Software Business Models

1. Product-Based Business Model

In a product-based business model, a company develops a software product and sells it to multiple customers. The same software is distributed to a large number of users without significant modifications. Since the product is developed once and sold many times, this model can be highly profitable when the software gains widespread acceptance.

For example, a company may develop accounting software for small businesses. Thousands of organisations can purchase and use the same software package. The company generates revenue from product sales, upgrades, and support services.

One advantage of this model is scalability. Once the software has been developed, additional copies can be distributed with relatively low cost. However, companies must continually improve the product to remain competitive and meet changing customer needs.

Imagine a textbook publisher that prints thousands of copies of the same book and sells them to students across different schools. Similarly, a software company develops one product and distributes it to many customers.

2. Service-Based Business Model

In a service-based business model, software companies develop customised solutions for specific clients. Instead of selling a standard product, the company provides software development, consulting, maintenance, and support services according to customer requirements.

SGOU

This model is common in software consulting firms. Clients approach the company with unique business needs, and developers create software tailored to those requirements. Since each project is different, a high level of interaction between the client and the development team is required.

The primary source of revenue comes from project contracts, development fees, maintenance agreements, and consulting services.

Suppose a hospital requires a specialised patient management system that includes features unique to its operations. A software company develops the system specifically for that hospital. The solution may not be suitable for other hospitals without modification, making it a service based project.

3. Software as a Service Business Model

The Software as a Service model has become increasingly popular with the growth of cloud computing. In this model, software is hosted on remote servers and accessed through the internet. Users do not need to install the software on their own computers. Instead, they access it using a web browser or mobile application.

Customers typically pay a monthly or annual subscription fee. This provides the software company with a continuous stream of revenue while allowing users to receive regular updates and new features without additional installation.

The Software as a Service model benefits both providers and customers. Organisations can deploy updates centrally, while users can access the software from different devices and locations.

Consider an online learning platform. Students can log in from their homes, access learning materials, attend virtual classes, and track progress without installing complex software. The service is delivered entirely through the internet.

4. Open-Source Business Model

The open-source business model allows software source code to be publicly available. Users can study, modify, and distribute the software according to the terms of the license. At first glance, it may seem difficult to earn money from software that is freely available, but many successful organisations have demonstrated otherwise.

Companies using this model often generate revenue through professional support services, consulting, training programs, premium features, and enterprise editions. The open-source approach encourages collaboration among developers and promotes innovation.

A major advantage of open-source software is transparency. Because the source code is available for inspection, security issues and software defects can often be identified and corrected by the community.

Imagine a community garden where everyone contributes effort and shares ideas to improve the space. Although the garden belongs to everyone, experts may still offer specialised services such as landscaping or maintenance for a fee. Open-source software works in a similar manner.

SGOU

Comparison of Software Business Models

Each business model has strengths and challenges. Product-based models focus on mass distribution, service-based models emphasise customisation, SaaS models provide recurring revenue, and open-source models encourage community participation. Organisations choose a business model based on factors such as target customers, market demand, available resources, and long-term business objectives.

For example, a startup may initially adopt a SaaS model because it provides predictable recurring income. On the other hand, a consulting company may prefer a service-based model because it specialises in custom software development for clients.

A software business model defines how a software organisation creates value, delivers solutions, and generates revenue. It provides a framework for business operations and strategic planning. Common software business models include product-based, service-based, Software as a Service , and open-source models. Understanding these models helps organisations select the most suitable approach for achieving both customer satisfaction and business profitability.

1.3.3 Revenue Models

Imagine that a group of software developers creates an educational application that helps students prepare for competitive examinations. The application contains video lectures, practice tests, performance analytics, and study materials. Thousands of students download the application and begin using its features. However, developing and maintaining such an application requires significant investment. The company must pay developers, purchase servers, maintain databases, provide customer support, and continuously improve the software. Therefore, the company needs a method to generate income from its software product.

This method of earning money is known as a revenue model. A revenue model plays a vital role in determining the financial success of a software business. Even if software is technologically advanced and highly useful, the organisation may struggle to survive without a proper revenue-generating strategy.

A Revenue Model is a framework that describes how an organisation generates income from its products, services, or business activities. In the software industry, it defines the methods through which a company earns money from software products and services.

In simple terms, a revenue model answers the question:

"How does the company make money from its software?"

The revenue model directly influences business growth, customer acquisition strategies, pricing decisions, and long-term sustainability.

1.3.3.1 Need for Revenue Models

Software development involves considerable costs. Organisations invest money in research, development, testing, marketing, maintenance, infrastructure, and technical

SGOU

support. Without a reliable source of income, it becomes difficult to sustain these activities.

A well designed revenue model helps companies recover development costs and generate profits. It also provides financial stability and supports future innovation. Companies that choose suitable revenue models can invest in new technologies, improve software quality, and expand their market reach.

Revenue models are particularly important in today's competitive software market, where customers expect regular updates, enhanced security, and continuous feature improvements.

1.3.3.2 Types of Revenue Model

1. Subscription Revenue Model

In the subscription revenue model, customers pay a recurring fee (monthly or yearly) to access software or services. They continue receiving updates, new features, and technical support as long as the subscription is active. This model provides a steady and predictable income for the company.

Example: Online learning platforms and cloud-based software.

2. One Time Purchase Revenue Model

In the one time purchase revenue model, customers pay once to buy the software and can use it permanently without recurring payments. The company earns revenue at the time of sale, while future income depends on new customers or software upgrades. Example: Desktop software applications.

3. Advertising Revenue Model

In the advertising revenue model, users can access the software for free, while the company earns money by displaying advertisements. The more users the platform attracts, the higher the advertising revenue.

Example: Social media platforms, search engines, and free mobile games.

4. Freemium Revenue Model

The freemium revenue model offers basic features free of charge while advanced or premium features require payment. Users can try the software before deciding to upgrade, helping companies attract a large customer base.

Example: Music streaming and productivity applications.

5. Transaction-Based Revenue Model

In the transaction-based revenue model, the company earns a commission or service fee whenever a transaction is completed through its platform. Revenue increases as the number of transactions grows.

SGOU

Examples: E-commerce websites, online marketplaces, payment gateways, and ticket booking systems.

A revenue model defines how a software company earns money from its products and services. It is a critical component of business strategy because it determines financial sustainability and growth potential. Common revenue models include subscription, one time purchase, advertising, freemium, and transaction-based approaches. Selecting an appropriate revenue model helps organisations balance customer value with business profitability and ensures long term success in the software industry.

1.3.4 Licensing Strategies

Suppose you purchase a movie ticket for a cinema. Buying the ticket gives you the right to watch the movie, but it does not allow you to copy the movie and sell it to others. The ticket specifies what you are allowed to do and what you are not allowed to do.

Software works in a similar way. When users purchase, download, or subscribe to software, they do not automatically gain complete ownership of it. Instead, they receive certain rights defined by a software license. These rights determine how the software can be used, copied, modified, distributed, or shared.

As software became a valuable business asset, organisations needed a systematic way to protect their intellectual property while allowing customers to use their products legally. This need led to the development of various licensing strategies.

A Licensing Strategy is the approach used by a software organisation to define the legal permissions, restrictions, and conditions associated with the use, modification, and distribution of its software.

In simple terms, a licensing strategy specifies:

- Who can use the software.
- How the software can be used.
- Whether users can modify the software.
- Whether users can distribute copies.
- What fees or subscriptions are required.

A software license acts as a legal agreement between the software provider and the user.

Why Licensing is Important

Software development requires substantial investment in terms of time, money, expertise, and resources. Without legal protection, anyone could copy and distribute software without compensating the developers who created it.

Licensing strategies help organisations protect their intellectual property and ensure fair compensation for their efforts. At the same time, licenses provide customers with clear guidelines regarding software usage.

SGOU

Licensing also promotes trust between software providers and users. Customers know exactly what rights they possess, while organisations can enforce usage policies and prevent unauthorised distribution.

Furthermore, licensing strategies influence business models, revenue generation, software maintenance, and customer relationships.

1.3.4.1 Types of Licensing Strategies

1. Proprietary Licensing Strategy

In the proprietary licensing strategy, the software remains the property of the developer, and users are only granted permission to use it. The source code is not available, so users cannot modify or redistribute the software. This strategy protects intellectual property and generates revenue through software sales.

Example: Commercial software like Microsoft Office.

2. Open-Source Licensing Strategy

Open-source licensing allows users to access, modify, and distribute the software's source code according to the license terms. It encourages collaboration, innovation, and transparency while allowing organisations to customise the software to their needs.

Example: Linux operating system.

3. Freeware Licensing Strategy

Freeware is software that users can download and use free of charge. However, the source code is usually not available, so users cannot modify it. Companies often use freeware to attract users and increase brand recognition.

Example: Adobe Acrobat Reader.

4. Subscription Licensing Strategy

In the subscription licensing strategy, users pay a recurring fee (monthly or yearly) to access the software. As long as the subscription is active, users receive updates, security patches, and technical support.

Examples: Microsoft 365 and Adobe Creative Cloud.

5. Trial Licensing Strategy

A trial licensing strategy provides software for a limited time or with limited features so users can evaluate it before purchasing. It helps customers understand the software's capabilities and make informed buying decisions.

Example: Free trial versions of antivirus or video editing software.

SGOU

1.3.4.2 Factors Influencing Licensing Strategy Selection

Organisations choose licensing strategies based on several factors.

- The nature of the software product plays an important role. Enterprise software often uses proprietary or subscription licensing, while community-driven projects frequently adopt open-source licensing.
- Market competition also influences licensing decisions. Companies may offer freeware or trial versions to attract customers in highly competitive markets.
- Revenue objectives, customer expectations, maintenance requirements, and legal considerations further affect licensing strategy selection.
- A successful licensing strategy balances customer accessibility with organisational profitability.

A licensing strategy defines the legal framework governing software usage, distribution, and modification. It establishes the relationship between software providers and users while protecting intellectual property rights. Common licensing strategies include proprietary, open source, freeware, subscription, and trial licensing. Each approach offers unique advantages and challenges, making licensing an important strategic decision for software organisations.

1.3.5 The Five Dimensions of a Software Project

Imagine that a college decides to develop an Online Examination System for its students. The management wants the system to be completed before the next semester begins. They also want advanced features such as online registration, automated evaluation, result publication, and performance analytics. At the same time, they have a limited budget and only a small development team.

The project manager now faces several important questions:

- What features should be included?
- How much time is available?
- What budget can be allocated?
- How many people are required?
- How can quality be maintained?

The success of the project depends on balancing these factors effectively. These factors are known as the Five Dimensions of a Software Project. Every software project, regardless of size, is influenced by these dimensions. Understanding them helps project managers plan, monitor, and control projects more effectively.

SGOU

The Five Dimensions of a Software Project are the major factors that influence project planning, execution, and success. They help managers understand the limitations and requirements of a project and guide decision-making throughout the software development life cycle.

The five dimensions are:

1. Scope
2. Time
3. Cost
4. Quality
5. Resources

These dimensions are interconnected. A change in one dimension often affects the others. Therefore, project managers must carefully balance them to achieve project objectives.

1.3.5.1 Scope

Scope refers to the work that must be completed to deliver a software product. It defines what features, functions, services, and deliverables will be included in the project.

In simple terms, scope answers the question:

"What exactly will the software do?"

The scope acts as a boundary for the project. It helps stakeholders understand what is included and what is excluded.

Suppose a restaurant owner wants a software system to manage orders and billing.

The project requirements may include:

- Customer order management
- Billing generation
- Payment processing
- Inventory tracking

These features define the scope of the project.

If later the owner requests online food delivery functionality, the scope increases. As a result, additional development work becomes necessary.

Importance of Scope

A clearly defined scope helps prevent misunderstandings between customers and developers. It ensures that everyone involved in the project shares the same expectations.

SGOU

Poorly defined scope often leads to scope creep, a situation where additional requirements are continuously added during development. Scope creep can increase costs, delay delivery, and create quality problems.

Therefore, project managers spend considerable time documenting and approving project scope before development begins.

Challenges Related to Scope

One common challenge is changing customer requirements. As users gain a better understanding of their needs, they may request new features. While some changes are necessary, excessive modifications can negatively affect project schedules and budgets.

Project managers must evaluate each requested change carefully and determine its impact on the project.

1.3.5.2 Time

Time refers to the duration available for completing the software project. It includes schedules, deadlines, milestones, and delivery dates.

Every software project operates within a specific timeframe. Customers expect software to be delivered on time so that they can begin using it according to their business plans.

Consider a college planning to conduct online admissions for the upcoming academic year.

If the admission portal is not completed before admissions begin, students will be unable to register. Even if the software is excellent, delayed delivery reduces its usefulness.

Similarly, software projects must meet their deadlines to provide value.

Importance of Time

Time management is essential because delays often lead to financial losses and customer dissatisfaction.

Organisations may lose business opportunities if software is delivered too late. Competitors may introduce similar products first, reducing market advantage.

Effective scheduling helps teams complete activities in a logical sequence and monitor project progress.

Factors Affecting Project Time

Several factors influence project schedules:

- Project complexity
- Team experience
- Requirement changes

SGOU

- Technical difficulties
- Availability of resources

Project managers use planning tools such as Gantt charts and project schedules to track timelines and identify delays.

1.3.5.3 Cost

Cost refers to the financial resources required to complete a software project. Developing software involves many expenses, including salaries, hardware, software tools, infrastructure, testing activities, training, and maintenance. Every project operates within a budget, making cost management an important responsibility.

Imagine constructing a house.

Expenses may include:

- Building materials
- Labour charges
- Electrical work
- Furniture
- Interior design

Similarly, software projects require financial investment in people, technology, and infrastructure.

Components of Software Project Cost

1. **Employee Salaries :** Developers, testers, analysts, designers, and project managers must be compensated for their work. In most software projects, salaries represent the largest portion of the budget.

2. **Hardware and Infrastructure:** Organisations may need servers, computers, networking equipment, and cloud services to support development and deployment activities.

3. **Software Tools:** Development tools, testing frameworks, project management software, and database systems may require licensing fees.

4. **Training and Support:** Team members may require training to learn new technologies or methodologies.

5. **Importance of Cost Management**:Poor cost estimation can create significant problems. If expenses exceed the budget, the organisation may face financial difficulties or be forced to reduce project scope.

Effective cost management helps ensure that projects remain financially viable while delivering expected results.

SGOU

1.3.5.4 Quality

Quality refers to the degree to which software satisfies requirements and meets customer expectations.

A software product may be completed on time and within budget, but if it contains defects or performs poorly, the project cannot be considered successful.

Quality ensures that software is reliable, secure, efficient, and user-friendly.

Consider purchasing a smartphone.

Customers expect:

- Good performance
- Reliable operation
- Attractive design
- Long battery life

If the smartphone frequently crashes, customers will be dissatisfied regardless of its price. Software users have similar expectations regarding software quality.

Importance of Quality

High quality software improves customer satisfaction and reduces maintenance costs. It also enhances the organisation's reputation and increases customer trust.

For critical systems such as banking applications, healthcare systems, and aviation software, quality is especially important because failures can have serious consequences.

Maintaining Software Quality

Organisations maintain quality through:

- Quality assurance activities
- Software testing
- Code reviews
- Process improvements
- Continuous monitoring

Investing in quality during development often reduces future maintenance costs.

1.3.5.5 Resources

Resources include all people, tools, technologies, equipment, and facilities required to complete a software project. Without adequate resources, even well-planned projects may fail. Resources form the foundation upon which project activities are performed.

SGOU

Example:Imagine organising a large cultural festival.

Successful execution requires:

- Event organisers
- Volunteers
- Sound systems
- Venues
- Financial support

Similarly, software projects require a variety of resources to achieve their objectives.

Types of Resources

1. **Human Resources:** Human resources include software developers, testers, analysts, designers, project managers, and technical support staff.

2. The skills and experience of team members significantly influence project success.

3. **Technical Resources:** These include hardware systems, servers, development tools, databases, and networking equipment.

4. **Financial Resources:** Financial resources provide the budget needed to support project activities.

5. **Infrastructure Resources:** Office facilities, communication systems, cloud platforms, and work environments support project operations.

Importance of Resources

The availability and quality of resources directly affect project productivity. Skilled employees can solve problems efficiently, while advanced tools can improve development speed and quality.

Resource shortages often result in delays, increased costs, and reduced quality.

Relationship Among the Five Dimensions

The five dimensions do not operate independently. They are closely connected.

For example:

Suppose a customer requests additional features.

The increase in scope may require:

- More development time
- Additional budget
- More resources
- Extra testing efforts

SGOU

Similarly, reducing the project budget may force the organisation to reduce resources or scope.

Because these dimensions influence one another, project managers must evaluate the impact of every decision carefully.

Example of Interrelationship

Consider a software company developing an e-commerce website.

The customer suddenly requests:

- Mobile application support
- AI-based recommendations
- Advanced reporting features

These new requirements increase the project scope.

As a result:

- More developers may be needed (Resources).
- Additional development time becomes necessary (Time).
- Project expenses increase (Cost).
- More testing activities are required (Quality).

This example demonstrates how one dimension affects all others.

Why Understanding the Five Dimensions is Important

Understanding these dimensions helps project managers:

- Plan projects effectively.
- Allocate resources efficiently.
- Control project costs.
- Maintain quality standards.
- Meet customer expectations.
- Reduce project risks.
- Improve project success rates.

Organisations that carefully manage these dimensions are more likely to deliver successful software projects.

SGOU

The five dimensions of a software project are scope, Time, Cost, Quality, and Resources. These dimensions determine how a project is planned, executed, and controlled. Scope defines what the software will deliver, time determines when it must be completed, cost represents the financial investment, quality measures how well requirements are satisfied, and resources provide the people and tools needed for development. Since these dimensions are closely interconnected, project managers must balance them carefully to achieve project success.

1.3.6 Balancing Project Constraints

Imagine that a software company has received a project to develop an Online Banking System. The client wants the software to include advanced security features, a user-friendly interface, mobile banking support, and fast performance. At the same time, the client expects the project to be completed within four months and has allocated only a limited budget.

The project manager now faces a difficult challenge. If more features are added, additional development time and money will be required. If the budget is reduced, fewer developers may be available, which can delay the project. If the deadline is shortened, the development team may have less time for testing, which could affect software quality.

This situation illustrates the concept of project constraints. Every software project operates under certain limitations, and project managers must balance these limitations carefully to deliver successful software. Achieving the right balance among different project constraints is one of the most important responsibilities of software project management.

Project Constraints are the factors or limitations that influence the planning, execution, and completion of a software project. These constraints determine how the project is managed and directly affect its success.

In software engineering, the major project constraints are Scope, Time, Cost, Quality of Resources

These constraints are closely related to one another. A change in one constraint usually affects the others. Therefore, project managers must carefully balance these constraints to meet customer expectations while ensuring project success.

1.3.6.1 Strategies for Balancing Project Constraints

Project managers use several strategies to balance project constraints and ensure successful project completion.

1. Clearly Define the Project Scope

A well defined scope helps the development team understand exactly what needs to be delivered. Clearly documenting project requirements prevents unnecessary changes during development and reduces the chances of scope creep. This enables the project to stay on schedule and within budget.

SGOU

Example: Before developing a college management system, the project manager prepares a detailed list of features such as student registration, attendance, and examination management. Any new feature requested later is evaluated before being added.

2. Prioritise Requirements

Not all project requirements have the same level of importance. Essential features should be developed first, while less important features can be added in future releases if time and budget permit. Prioritisation helps ensure that the most valuable functions are completed on time.

Example: In an online shopping application, product search, shopping cart, and payment features are developed before adding product recommendations or chat support.

3. Plan Resources Effectively

Proper allocation of developers, testers, software tools, and infrastructure helps improve productivity and avoid unnecessary delays. Skilled resources should be assigned to critical tasks to achieve better results.

Example: Experienced developers are assigned to develop the payment module of an online banking application because it requires high security and accuracy.

4. Monitor Project Progress Continuously

Regular monitoring helps identify delays, budget overruns, and quality issues at an early stage. Project managers can then take corrective actions before problems become serious.

Example: Weekly project review meetings help the team track completed tasks, identify risks, and adjust schedules if necessary.

5. Communicate with Stakeholders

Effective communication among customers, developers, testers, and project managers ensures that everyone understands project goals and any changes in requirements. Good communication helps avoid misunderstandings and supports better decision-making.

Example**:** If a customer requests additional features, the project manager discusses the impact on cost and delivery time before approving the changes.

6. Manage Risks Proactively

Potential risks should be identified and addressed early in the project. Preparing contingency plans helps minimise the impact of unexpected problems such as technical issues, resource shortages, or changing customer requirements.

Example: A project manager keeps backup developers available in case a key team member becomes unavailable during development.

SGOU

Importance of Balancing Project Constraints

Balancing project constraints helps organisations deliver software that meets customer requirements while staying within the available time and budget. It improves project planning, resource utilisation, software quality, and customer satisfaction. Proper balancing also reduces project risks, avoids unnecessary delays, and increases the liklihood of successful project completion.

Every software project operates under constraints such as scope, time, cost, quality, and resources. These constraints are interdependent, meaning that changes in one constraint usually affect the others. Therefore, project managers must carefully balance these factors by defining project scope, prioritising requirements, allocating resources efficiently, monitoring progress, communicating effectively, and managing risks. Successful balancing of project constraints helps organisations deliver high-quality software on time, within budget, and according to customer expectations.

Recap

- A software business model explains how software organisations create value, deliver products, and earn revenue.
- Common business models include product-based, service-based, SaaS, and open-source models.
- Revenue models determine how organisations generate income through subscription, advertising, freemium, one-time purchase, and transaction-based approaches.
- Licensing strategies define the legal rights and restrictions for software usage, modification, and distribution.
- The five dimensions of a software project are Scope, Time, Cost, Quality, and Resources.
- These project dimensions are interdependent, and changes in one dimension affect the others.
- Balancing project constraints helps organisations deliver software within budget, schedule, and quality expectations.
- Effective project planning and business strategies improve software quality, customer satisfaction, and organisational success.

Objective Type Questions

1. Which business model delivers software through the Internet?
2. Which revenue model charges customers monthly or yearly?

SGOU

3. Which licensing strategy provides access to the source code?
4. Which project dimension defines the work to be completed?
5. Which project dimension represents the project budget?
6. Which project dimension measures customer satisfaction and software excellence?
7. Which project dimension includes developers, testers, and infrastructure?
8. Which software business model develops customised software for clients?
9. Which licensing strategy provides software free for evaluation?
10. Which revenue model earns money by displaying advertisements?
11. What is the recurring payment model called?
12. Which project management problem occurs when new requirements are continuously added?
13. Which project dimension specifies the project deadline?
14. Which licensing strategy protects proprietary source code?
15. Which business model encourages community collaboration?

Answers to Objective Type Questions

1. SaaS.
2. Subscription.
3. Open-source.
4. Scope.
5. Cost.
6. Quality.
7. Resources.
8. Service-based.
9. Trial.
10. Advertising.
11. Subscription.
12. Scope Creep.
13. Time.
14. Proprietary.
15. Open-source.

SGOU

Assignments

1. Explain the concept of a Software Business Model. Discuss the different types of software business models with suitable examples.

2. Define a Revenue Model. Explain the various revenue models used by software organisations and discuss their advantages.

3. What is a Licensing Strategy? Explain the different licensing strategies adopted in software development with suitable examples.

4. Describe the Five Dimensions of a Software Project. Explain how each dimension influences project success.

5. Discuss the importance of balancing project constraints in software engineering. Explain various strategies used by project managers.

6. Explain the relationship between Scope, Time, Cost, Quality, and Resources with suitable examples.

7. Compare different software business models and licensing strategies. Discuss how they contribute to business sustainability and customer satisfaction.

Suggested Reading

1. Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Addison-Wesley.

2. Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley.

3. McConnell, S. (2004). *Code complete* (2nd ed.). Microsoft Press.

4. Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

Reference

1. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill.
2. Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
3. Schwalbe, K. (2022). *Information technology project management* (10th ed.). Cengage Learning.
4. Project Management Institute. (2021). *A guide to the project management body of knowledge (PMBOK® Guide)* (7th ed.). Project Management Institute.

SGOU

UNIT 4

Software Ethics and Quality Management

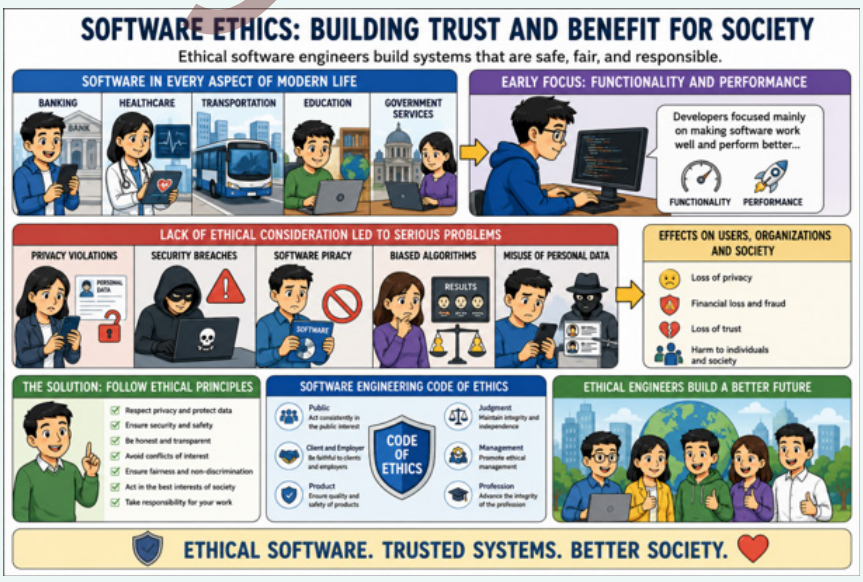
Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept and importance of software engineering ethics
- describe the responsibilities of software engineers toward society, clients, and employers
- understand the principles of the Software Engineering Code of Ethics
- apply ethical decision making in real world software development situations

Prerequisite

Software has become a part of almost every aspect of modern life, including banking, healthcare, transportation, education, and government services. As software systems became more powerful, developers initially focused mainly on functionality and performance, often paying little attention to ethical responsibilities. This led to problems such as privacy violations, security breaches, software piracy, biased algorithms, and misuse of personal data. These issues affected users, organisations, and society, reducing trust in software systems.



SGOU

To overcome these challenges, software engineers must follow ethical principles while developing and maintaining software. The Software Engineering Code of Ethics provides professional guidelines that help engineers make responsible decisions, protect user privacy, ensure software quality, and act in the best interests of society. Studying software ethics enables future software professionals to understand their responsibilities, avoid unethical practices, and develop software that is safe, reliable, and socially beneficial.

Keywords

Software Ethics, Professional Responsibility, Integrity, Confidentiality, Privacy, Accountability, Public Interest, Code of Ethics

Discussion

1.4.1 Introduction

Imagine that you have booked a train ticket online for your semester vacation. When you reach the railway station, the ticket examiner informs you that the same seat has been allotted to another passenger due to a software error. Both passengers become confused, the railway staff spend time resolving the issue, and the journey begins with unnecessary inconvenience. Although the error may seem small, it affects many people and reduces their confidence in the software system.

Now consider another situation. A hospital uses software to store patients' medical records. During the final testing phase, a software engineer discovers that unauthorised users can access confidential patient information. Correcting the problem will delay the software release, but ignoring it may expose sensitive medical records and violate patient privacy. What should the engineer do? Should the software be released on time, or should the release be postponed until the problem is fixed?

These situations show that software engineering is not only about writing programs but also about making responsible decisions. Every software engineer must ensure that the software they develop is accurate, reliable, secure, and beneficial to society. A program may function correctly, but if it ignores ethical principles or fails to meet quality standards, it can still cause financial loss, security breaches, or harm to users.

Today, software has become an essential part of everyday life. It is used in banking, healthcare, education, transportation, e-commerce, communication, entertainment, and industrial automation. Millions of people rely on software every day to perform important tasks. As a result, software engineers have a professional responsibility to develop software that users can trust.

To meet this responsibility, software engineering focuses on two closely related concepts: software ethics and software quality management. Software ethics helps software professionals make responsible and morally correct decisions during software

SGOU

development. Software quality management ensures that the software satisfies user requirements and meets established quality standards. Together, these concepts help organisations develop software that is reliable, secure, efficient, and user-friendly. Fig 1.4.1 shows software supports almost every aspect of modern life. Therefore, software engineers must develop systems responsibly and maintain high-quality standards.

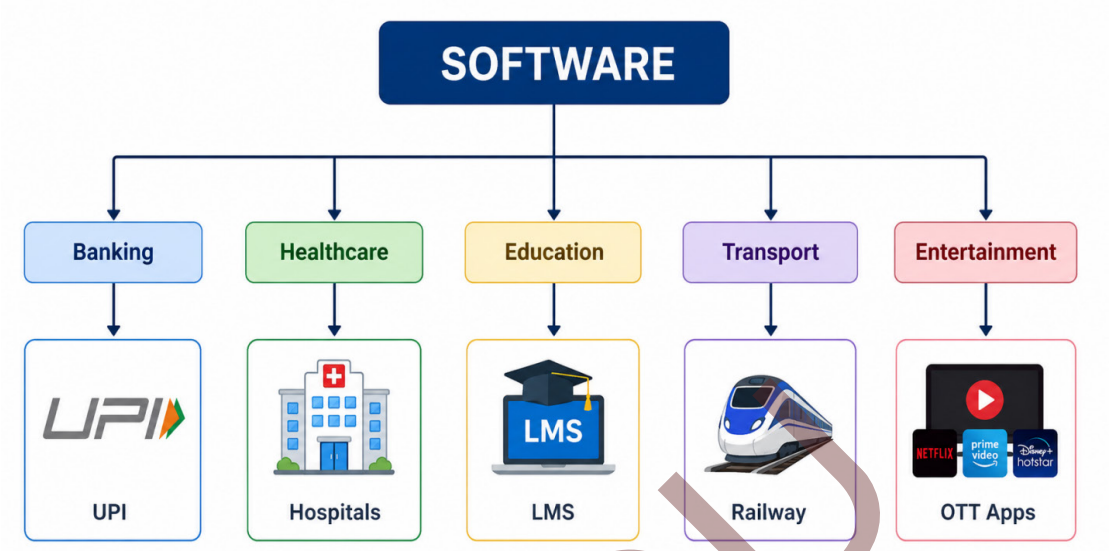


Fig 1.4.1 Software in Everyday Life

Relationship Between Ethics and Quality

Software ethics and software quality management are closely connected. Ethical software engineers are committed to honesty, fairness, and responsibility, while quality management provides the processes and standards needed to develop dependable software. When ethical practices are followed along with effective quality management, organisations can deliver software that satisfies users, protects society, and builds public trust.

Good software is not only functional but also ethical, reliable, secure, and user-friendly. Ethics guides responsible decisions, while quality management ensures those decisions result in dependable software.

1.4.2 Software Ethics

Every profession follows certain moral principles that guide the behaviour of its members. For example, doctors are expected to protect patient confidentiality, teachers should evaluate students fairly, and lawyers must represent their clients honestly. Similarly, software engineers are expected to develop software responsibly and in a way that benefits users and society.

The word ethics refers to the principles and values that help people distinguish between right and wrong behaviour. Ethics is not limited to following laws; it also reflects an individual's honesty, responsibility, fairness, and professional integrity. Ethical behaviour encourages people to make decisions that are morally correct, even when they face pressure or difficult situations.

SGOU

In software engineering, ethical behaviour influences every stage of software development, from gathering user requirements to designing, testing, deploying, and maintaining software. Ethical software engineers always consider how their decisions may affect users, organisations, and society.

Software ethics is the branch of professional ethics that deals with the moral principles and responsibilities followed by software engineers during the design, development, testing, deployment, and maintenance of software systems.

1.4.2.1 Why Do Software Engineers Need Ethics?

Software is used in almost every aspect of modern life, including banking, healthcare, transportation, education, online shopping, and government services. A single software error can affect thousands or even millions of users. Therefore, software engineers must think beyond writing correct code and consider the social and ethical impact of their work.

For example, suppose a software engineer discovers a security weakness in an online banking application. If the engineer ignores the problem to meet the project deadline, hackers may exploit the vulnerability and steal customers' money. An ethical software engineer, however, reports the issue immediately and ensures that it is corrected before the software is released. Although this may delay the project, it protects users and maintains public trust.

Similarly, consider an online examination system. During testing, engineers discover that students can access question papers before the examination begins due to a software defect. Releasing such software would make the examination unfair. Ethical software professionals understand that protecting users and society is more important than meeting deadlines or reducing development costs.

These examples show that ethical decisions help prevent financial losses, protect user privacy, improve software reliability, and maintain the reputation of software organisations.

SGOU

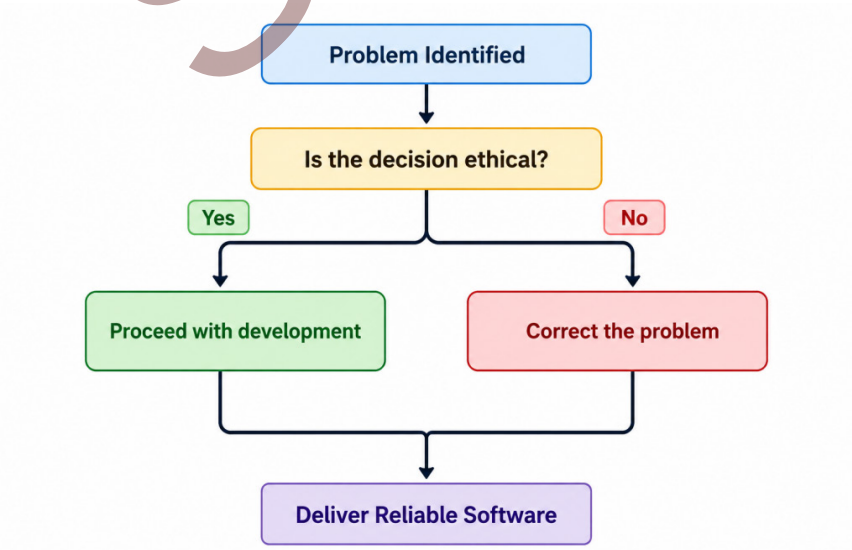


Fig 1.4.2 Ethical Decision-Making Process

1.4.2.2 Characteristics of an Ethical Software Engineer

An ethical software engineer demonstrates qualities that promote trust, responsibility, and professionalism. These qualities not only improve software quality but also strengthen the reputation of the organisation.

1. **Honesty:** Ethical software engineers provide truthful information about software performance, project progress, and identified defects. They do not hide errors or make false promises to clients simply to complete a project on time.

2. **Responsibility:** They take responsibility for the software they develop and understand that their decisions may affect many users. They carefully test software and ensure that important defects are corrected before release.

3. **Respect for Privacy:** Software often stores personal information such as passwords, medical records, and financial details. Ethical engineers protect this information from unauthorised access and maintain user confidentiality.

4. **Respect for Intellectual Property:** Ethical professionals respect copyrights, software licenses, patents, and trademarks. They do not copy software, documents, or source code without proper permission.

5. **Professional Competence**: Technology changes rapidly. Ethical software engineers continuously improve their knowledge by learning new programming languages, development tools, security practices, and quality standards.

6. **Fairness and Teamwork:** Software development is usually a team effort. Ethical engineers respect colleagues, share knowledge, provide constructive feedback, and avoid discrimination or unfair treatment.

1.4.2.3 Importance of Software Ethics

Software ethics plays a vital role in ensuring that technology benefits both individuals and society. Ethical behaviour helps software engineers develop systems that are reliable, secure, and trustworthy. It also reduces the chances of software failures, legal disputes, and security breaches.

Organisations that encourage ethical practices gain the confidence of customers because users know that their information is handled responsibly. Ethical behaviour also improves teamwork, promotes transparency, and helps maintain the professional reputation of software engineers.

With the increasing use of artificial intelligence, cloud computing, mobile applications, and the Internet of Things (IoT), software ethics has become more important than ever. Developers must carefully consider issues such as data privacy, cybersecurity, fairness, and the responsible use of technology.

A software company developed a mobile health application that stored patients' medical records. During security testing, an engineer discovered that unauthorised users could access confidential information due to a programming error. Although fixing the

SGOU

problem delayed the software release by a few days, the company decided to correct the defect before launching the application. As a result, patient information remained secure, legal issues were avoided, and customers trusted the application.

This example shows that ethical decisions contribute directly to software quality and public safety.

> Ethics helps software engineers make responsible decisions. Ethical behaviour protects users, improves software quality, strengthens public trust, and ensures that technology benefits society.

Understanding software ethics is the first step toward becoming a responsible software professional. To ensure that ethical behaviour is consistently followed across the software industry, professional organisations have established a common set of guidelines known as the Software Engineering Code of Ethics. The next section explains these ethical principles in detail.

1.4.3 The Software Engineering Code of Ethics

Imagine that a software company is developing an online banking application. During the final stage of testing, the development team discovers a defect that may occasionally duplicate money transfers. Correcting the problem requires additional testing, which will delay the project. The management is under pressure to release the software on the promised date.

What should the software engineers do?

Should they release the software to meet the deadline, or should they postpone the release to protect customers from financial loss?

Situations like this are common in software development. To help software professionals make responsible decisions, internationally accepted ethical guidelines have been established. These guidelines are known as the Software Engineering Code of Ethics. They encourage software engineers to place public welfare, honesty, quality, and professional responsibility above personal or organisational interests.

The Software Engineering Code of Ethics is a set of professional principles that guides software engineers to develop software responsibly, honestly, and in the best interests of users, clients, employers, and society.

1.4.3.1 Why is the Code of Ethics Important?

Software systems control many critical services such as banking, healthcare, transportation, education, and communication. Errors or unethical decisions in these systems can lead to financial losses, privacy violations, or even threats to human life.

The Code of Ethics helps software engineers make fair and responsible decisions throughout the software development process. It promotes public trust, improves software quality, protects users, and enhances the reputation of the software engineering profession.

SGOU

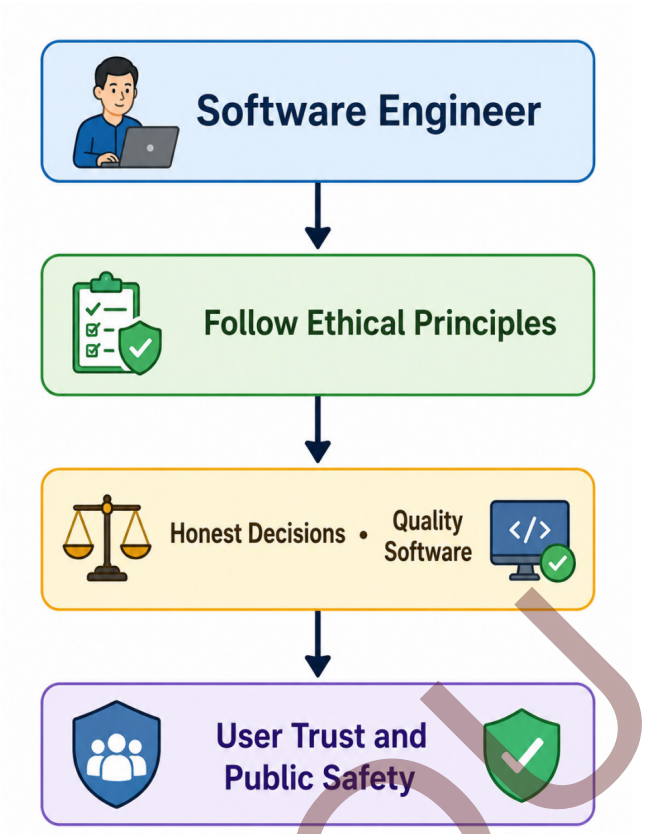


Fig 1.4.3 Purpose of the Software Engineering Code of Ethics

1.4.3.2 Principles of the Software Engineering Code of Ethics

The ACM (Association for Computing Machinery) and the IEEE Computer Society jointly developed the Software Engineering Code of Ethics and Professional Practice. It consists of eight principles that guide software engineers in their professional work.

1. Public

The first principle states that software engineers should always act in the best interests of the public. The safety, health, and welfare of society must always receive the highest priority.

Software engineers should avoid releasing software that may harm users or create security risks. If a defect could seriously affect people, they should report it immediately and ensure that it is corrected before deployment.

Example

A software engineer discovers a defect in hospital management software that could display incorrect patient information. Instead of ignoring the issue to meet the deadline, the engineer reports the defect and postpones the release until it is fixed.

2. Client and Employer

Software engineers should serve the interests of their clients and employers honestly and responsibly while ensuring that public welfare is not compromised.

SGOU

They should protect confidential information, provide accurate project reports, and avoid conflicts of interest. At the same time, they should refuse requests that may harm users or violate ethical standards.

Example

A client asks a developer to collect users' personal data without informing them. The engineer explains that this violates privacy principles and recommends obtaining proper user consent.

3. Product

Software engineers should ensure that every software product meets high professional standards of quality. The software should be reliable, secure, thoroughly tested, and properly documented before it is released.

Quality should never be sacrificed simply to reduce development costs or meet unrealistic deadlines.

Example

Before releasing an online shopping application, developers perform additional security testing to remove payment-related vulnerabilities, ensuring safe transactions for customers.

4. Judgment

Software engineers should make professional decisions based on technical knowledge, facts, and ethical responsibility. Their judgment should remain honest, independent, and free from personal or financial pressure.

They should provide truthful advice to clients and clearly communicate software limitations or risks.

Example

A software engineer recommends a more secure database system for a hospital application, even though it requires additional development effort, because it better protects patient information.

5. Management

Managers and project leaders should promote ethical software development by creating an environment that supports quality, teamwork, and responsible decision-making.

They should provide adequate time for planning, testing, reviews, and quality assurance instead of encouraging shortcuts that may reduce software quality.

Example

A project manager extends the testing schedule after discovering several critical defects rather than releasing unreliable software.

SGOU

6. Profession

Software engineers should maintain the dignity and reputation of the software engineering profession. They should follow professional standards, respect laws, and encourage ethical practices within their organisations.

Sharing knowledge, supporting innovation, and maintaining professional integrity strengthen public confidence in software engineering.

Example

An experienced software engineer conducts training sessions on secure coding practices for new employees.

7. Colleagues

Software engineers should treat colleagues fairly, respectfully, and professionally. They should encourage teamwork, share knowledge, and provide constructive feedback.

An ethical work environment improves communication, increases productivity, and leads to higher software quality.

Example

During a code review, a senior developer patiently explains programming mistakes to a junior developer and helps improve the quality of the software instead of criticising them.

8. Self

Technology changes rapidly, so software engineers should continuously improve their technical knowledge and ethical understanding. Lifelong learning enables them to develop modern, secure, and high-quality software.

Engineers should attend training programmes, learn new technologies, and stay updated with emerging software engineering practices.

Example

A software engineer learns cloud security techniques before working on a cloud-based application to ensure better protection of user data.

SGOU

Table 1.4.1 Eight Principles

Principle	Main Responsibility	Example
Public	Protect society and public welfare	Do not release unsafe software
Client and Employer	Serve clients honestly	Protect confidential information
Product	Develop high-quality software	Perform thorough testing
Judgment	Make honest technical decisions	Report software risks truthfully
Management	Promote ethical project management	Allow sufficient testing time
Profession	Maintain professional reputation	Follow standards and laws
Colleagues	Respect and support team members	Share knowledge and cooperate
Self	Continue professional learning	Learn new technologies

Case Study

A company was developing an online examination system. During testing, engineers found that students could access examination questions by modifying the website address (URL). Although fixing the issue delayed the project, the company corrected the defect before release. This decision protected the fairness of the examination process and increased users' trust in the software.

This case demonstrates that following the Software Engineering Code of Ethics helps organisations develop secure, reliable, and trustworthy software.

The Software Engineering Code of Ethics reminds software engineers that public welfare, honesty, quality, and professional responsibility should always come before personal or financial gain.

Ethical software engineers strive to develop software responsibly, but ethical behaviour alone is not enough. Software must also satisfy users' expectations for reliability, security, performance, and usability. In the next section, we will study Software Quality Requirements and the characteristics that define high-quality software.

1.4.4 Software Quality Requirements

Suppose you want to purchase a new smartphone. Before buying it, you compare different models based on battery life, speed, camera quality, storage capacity, security features, and ease of use. Although many smartphones can perform similar basic functions, users usually choose the one that offers better quality and a better user experience.

SGOU

The same principle applies to software. Two applications may perform the same task, but users always prefer software that is reliable, secure, easy to use, and responds quickly. For example, two online banking applications may allow users to transfer money and check account balances. However, if one application frequently crashes or takes too long to respond, users will lose confidence in it.

Therefore, software quality is determined not only by what the software does, but also by how well it performs its functions. To achieve this, software developers identify quality requirements during the early stages of software development.

Software quality requirements are the characteristics and standards that a software product must satisfy to meet user expectations and perform efficiently, reliably, and securely.

1.4.4.1 Importance of Software Quality Requirements

Software quality requirements help developers understand the expected level of quality before development begins. Clearly defined quality requirements reduce misunderstandings between clients and developers and provide a common goal for the entire project team.

Quality requirements also make software testing easier because testers know exactly what standards the software should meet. As a result, defects can be identified and corrected early, reducing maintenance costs and improving customer satisfaction.

In today's competitive software industry, organisations that deliver high-quality software gain greater customer trust and improve their market reputation.

> Software that performs the required functions but fails to meet quality expectations may still be considered unsuccessful by users.

1.4.4.2 Functional and Non-Functional Requirements

Software requirements are generally divided into functional requirements and non-functional requirements.

a. Functional Requirements

Functional requirements describe what the software should do. They specify the services and operations that the system must perform to satisfy user needs.

For example, an online banking system should allow users to:

- Create an account
- Transfer money
- Check account balance
- Download account statements

These functions define the main purpose of the software.

SGOU

b. Non-Functional Requirements

Non-functional requirements describe how well the software should perform. They focus on quality characteristics such as performance, security, reliability, and usability.

For example, an online banking application should:

- Complete transactions within a few seconds.
- Protect customer data from unauthorised access.
- Remain available 24 hours a day.
- Support thousands of users simultaneously.

Although these requirements do not add new functions, they greatly influence user satisfaction.

Table 1.4.2 Functional vs. Non-Functional Requirements

Functional Requirements	Non-Functional Requirements
Describe what the software should do.	Describe how well the software should perform.
Related to system functions.	Related to software quality.
Example: Login, payment, report generation.	Example: Security, reliability, performance.
Verified through functional testing.	Verified through performance and security testing.

1.4.4.3 Software Quality Attributes

Software quality is measured using several important characteristics called quality attributes. These attributes help determine whether software meets user expectations.



Fig 1.4.4 Major Software Quality Attributes

SGOU

1. Reliability

Reliability is the ability of software to perform its intended functions accurately without failure for a specified period.

Example: An online banking system processes thousands of transactions every day without crashing or losing customer data.

2. Usability

Usability refers to how easy it is for users to learn and operate the software.

Software with simple navigation, clear instructions, and user-friendly interfaces improves user satisfaction.

Example: A mobile payment application with large icons and simple menus allows users of all ages to make payments easily.

3. Efficiency

Efficiency is the ability of software to perform tasks quickly while using minimum system resources such as memory, processing time, and storage.

Example: A search engine displays relevant search results within a few seconds.

4. Security

Security protects software and user data from unauthorized access, modification, or misuse.

Secure software uses techniques such as passwords, encryption, and authentication to safeguard information.

Example: Internet banking applications use passwords and One-Time Passwords (OTP) to verify users before completing transactions.

5. Maintainability

Maintainability refers to the ease with which software can be corrected, modified, or enhanced after deployment.

Well-organised code and proper documentation make maintenance easier.

Example: A college management system can be easily updated when new courses are introduced.

6. Portability

Portability is the ability of software to run on different hardware platforms or operating systems with little or no modification.

Example: A web application that works on Windows, Linux, and macOS.

SGOU

7. Scalability

Scalability refers to the ability of software to handle increasing numbers of users or larger amounts of data without affecting performance.

Example: An e-commerce website successfully manages heavy customer traffic during festive sales.

8. Availability

Availability indicates how often software remains operational and accessible whenever users need it.

Example: ATM services are expected to be available 24 hours a day.

9. Reusability

Reusability is the ability to use existing software components in different applications, reducing development time and cost.

Example: A login module developed for one application can also be used in another application.

Table 1.4.3 Major Software Quality Attributes

Quality Attribute	Meaning	Example
Reliability	Performs correctly without failure	Banking software
Usability	Easy to learn and use	Mobile payment app
Efficiency	Uses minimum resources	Search engine
Security	Protects data from unauthorised access	Internet banking
Maintainability	Easy to modify	College management system
Portability	Runs on multiple platforms	Web application
Scalability	Handles increasing workload	E-commerce website
Availability	Accessible whenever required	ATM system
Reusability	Components can be reused	Login module

A company developed a food delivery application that provided all the required features, such as placing orders and online payment. However, during peak hours the application became slow and frequently crashed. Although the functional requirements were satisfied, the software failed to meet quality requirements such as reliability and efficiency. After improving server capacity and optimising the software, the application became faster and more reliable, resulting in increased customer satisfaction.

SGOU

Functional requirements define what the software should do, while non-functional requirements define how well it should perform. Both are essential for developing high-quality software.

After identifying software quality requirements, organisations must ensure that these requirements are achieved throughout the software development process. This is accomplished through a Software Quality Plan, which defines the quality objectives, standards, testing activities, and responsibilities for the project. The next section discusses the Software Quality Plan in detail.

1.4.5 Software Quality Plan

Imagine that a construction company is building a new hospital. Before construction begins, the company prepares a detailed plan that specifies the quality of materials, safety measures, inspection procedures, and responsibilities of engineers. This plan helps ensure that the building is safe, durable, and meets the required standards.

Software development follows a similar approach. Developing software without planning quality may result in defects, security vulnerabilities, poor performance, and increased maintenance costs. Therefore, software organisations prepare a Software Quality Plan (SQP) before development begins. This plan ensures that quality is maintained throughout the Software Development Life Cycle (SDLC), from requirement analysis to software maintenance.

A Software Quality Plan serves as a roadmap for developers, testers, quality assurance engineers, and project managers. It specifies the quality objectives, standards, review procedures, testing methods, and responsibilities of team members. By following the plan, organisations can develop software that is reliable, secure, and meets customer expectations.

A Software Quality Plan (SQP) is a document that describes the quality objectives, standards, procedures, responsibilities, and activities required to ensure that a software product meets specified quality requirements.

1.4.5.1 Objectives of a Software Quality Plan

The main objective of a Software Quality Plan is to ensure that software meets customer requirements and organisational quality standards. It guides the development team in performing quality-related activities throughout the project.

The major objectives of a Software Quality Plan are:

- To define quality goals before software development begins.
- To ensure compliance with quality standards and organisational policies.
- To reduce software defects through systematic reviews and testing.
- To assign quality-related responsibilities to team members.

SGOU

- To improve customer satisfaction by delivering reliable software.
- To reduce project risks and maintenance costs.

1.4.5.2 Importance of a Software Quality Plan

A Software Quality Plan helps the project team work towards a common quality goal. It ensures that coding standards, testing procedures, and documentation practices are followed consistently throughout the project.

By identifying quality issues early, organisations can reduce the cost of correcting defects after software deployment. A well-prepared quality plan also improves communication among team members, increases customer confidence, and supports the successful completion of software projects.

> Quality cannot be added at the end of software development. It must be planned and maintained throughout the entire development process.

1.4.5.3 Components of a Software Quality Plan

Although the format of a Software Quality Plan may vary from one organisation to another, most plans include the following components.

1. Project Description

Provides a brief overview of the software project, including its purpose, scope, and major features.

2. Quality Objectives

Defines the quality goals that the software should achieve, such as reliability, security, performance, and usability.

3. Quality Standards

Specifies the standards, coding guidelines, and organisational policies to be followed during software development.

4. Roles and Responsibilities

Clearly identifies the responsibilities of developers, testers, quality assurance engineers, and project managers.

5. Review and Audit Procedures

Describes activities such as requirement reviews, design reviews, code reviews, and quality audits to identify defects at an early stage.

6. Testing Strategy

Specifies the testing methods to be used, such as unit testing, integration testing, system testing, and user acceptance testing.

SGOU

7. Documentation

Lists the documents to be prepared and maintained, including requirement specifications, design documents, test reports, and user manuals.

8. Risk Management

Identifies potential quality risks and defines strategies to minimise or eliminate them.



Fig 1.4.5 Components of a Software Quality Plan

A Software Quality Plan combines different quality activities to ensure the development of reliable software.

1.4.5.4 Software Quality Planning Process

The Software Quality Planning process begins by understanding customer requirements. Based on these requirements, quality objectives and standards are defined. The project team then performs reviews, testing, and quality assurance activities throughout software development. Finally, the software is evaluated to ensure that it satisfies the required quality standards before release.

Example: Software Quality Plan

SGOU

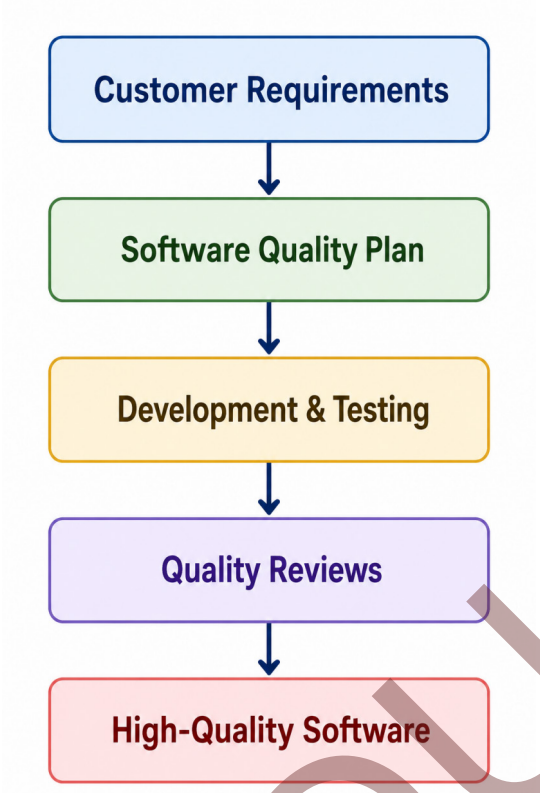


Fig 1.4.6 Software Quality Planning Process

Consider a university developing an Online Library Management System. Before development begins, the project team prepares a Software Quality Plan.

The plan specifies that:

- All modules must follow coding standards.
- Every module should undergo peer review.
- Unit and integration testing are mandatory.
- Student information must be protected through secure authentication.
- Software should respond to user requests within three seconds.
- Progress reviews should be conducted every two weeks.

By following this plan, the university can develop software that is secure, reliable, and easy to maintain.

1.4.5.6 Relationship Between Ethics and Quality Management

Software ethics and software quality management are closely related. Ethical software engineers develop software honestly and responsibly, while quality management provides the methods and standards required to achieve quality.

SGOU

For example, an ethical software engineer reports software defects instead of hiding them to meet project deadlines. The Software Quality Plan ensures that these defects are documented, corrected, tested, and verified before the software is released. Thus, ethics and quality management work together to produce software that is safe, reliable, and beneficial to society.

In this unit, you learned that software ethics guides software engineers in making responsible and professional decisions, while the Software Engineering Code of Ethics provides eight principles that promote public welfare, honesty, quality, and continuous professional development. You also studied software quality requirements, including functional and non-functional requirements, and the key quality attributes such as reliability, usability, security, and maintainability. Finally, you learned that a Software Quality Plan helps organisations plan, monitor, and maintain quality throughout the software development life cycle, resulting in software that satisfies users and meets professional standards.

Recap

- Software engineering involves both technical skills and ethical responsibilities.
- Ethical practices help develop reliable, secure, and trustworthy software.
- Software engineers must protect user privacy and confidential information.
- The Software Engineering Code of Ethics provides professional guidelines for ethical behaviour.
- Ethical decisions improve public trust and organisational reputation.
- Ignoring ethical principles can lead to software failures, legal issues, and financial losses.
- Engineers should always consider the impact of software on users and society.
- Professional responsibility is essential throughout the software development life cycle.

SGOU

Objective Type Questions

1. What is software ethics?
2. Which document provides professional guidelines for software engineers?
3. Who is responsible for protecting user privacy?
4. What should software engineers protect from unauthorised access?
5. Which principle focuses on honesty and truthfulness?
6. Which principle requires engineers to act in the public interest?
7. What is the unauthorised disclosure of information called?
8. Which quality should software engineers maintain while making decisions?
9. Which profession follows the Software Engineering Code of Ethics?
10. What should an engineer do if software contains a serious security flaw?
11. Which ethical principle protects confidential information?
12. What is the main purpose of software ethics?
13. Which sector stores confidential patient records?
14. Software engineers should prioritise public safety. (True/False)
15. Ethical software increases user trust. (True/False)

SGOU

Answers to Objective Type Questions

1. Moral principles followed in software development.
2. Software Engineering Code of Ethics.
3. Software Engineer.
4. User data.
5. Integrity.
6. Public Interest.
7. Privacy breach.
8. Professional integrity.

9. Software Engineering.
10. Report and fix the defect.
11. Confidentiality.
12. To ensure responsible software development.
13. Healthcare.
14. True.
15. True.

Assignments

1. Define software engineering ethics and explain its importance with suitable examples.
2. Discuss the need for ethical practices in software engineering.
3. Explain the Software Engineering Code of Ethics and its significance in software development.
4. Describe the responsibilities of software engineers toward clients, employers, and society.
5. Explain the ethical issues commonly faced by software engineers with suitable examples.
6. Discuss how ethical principles improve software quality and user trust.
7. Explain the consequences of ignoring software engineering ethics with real-life examples.

SGOU

Suggested Reading

1. Gotterbarn, D., Miller, K., & Rogerson, S.. (1999). *Software Engineering Code of Ethics*. ACM/IEEE Computer Society.
2. Spinello, R. A.. (2020). *Cyberethics: Morality and law in cyberspace* (7th ed.). Jones & Bartlett Learning.
3. Reynolds, G. W.. (2019). *Ethics in information technology* (6th ed.). Cengage Learning.
4. Johnson, D. G.. (2009). *Computer ethics* (4th ed.). Pearson.

Reference

1. Pressman, R. S., & Maxim, B. R.. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

2. Sommerville, I.. (2016). *Software engineering* (10th ed.). Pearson.

3. IEEE-CS/ACM Joint Task Force. (2018). *Software Engineering Code of Ethics and Professional Practice*. IEEE Computer Society & ACM.

4. Martin, R. C.. (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.

SGOU



BLOCK 2

Quality Management Standards and Models

SGOU



Software Quality Standards and Business Models

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept, objectives, and importance of Quality Management Standards in software organisations.
- describe the major quality management standards, including ISO 9001, ISO 14001, ISO 45001, ISO 27001, IATF 16949, and ISO 13485.
- apply the concept of Cost of Quality (CoQ) to identify prevention, appraisal, internal failure, and external failure costs in software development.
- analyse the differences among major Quality Management Standards based on their focus, application areas, and benefits.
- analyse different software business models and their influence on software quality management, customer satisfaction, and continuous improvement

Prerequisite

Consider a software company developing an online banking application used by thousands of customers every day. The application must process transactions accurately, protect sensitive customer information, and provide reliable performance. If proper quality practices are not followed, the software may contain errors, security vulnerabilities, or performance problems, leading to customer dissatisfaction and financial losses. To understand how such issues can be prevented and managed, learners should have a basic understanding of software engineering, software development, software testing, software quality, customer requirements, and project management. Quality Management Standards help organisations establish systematic processes to develop reliable software, reduce defects, satisfy customers, and improve their overall performance. .

This unit is relevant because it helps learners understand how organisations manage and improve software quality. It introduces major Quality Management Standards, Cost of Quality, and different software business models. Learners can understand the importance of preventing defects, testing software, reducing failure costs, improving customer satisfaction, and achieving continuous improvement. The knowledge gained

SGOU

from this unit helps learners understand how effective quality management can improve software reliability, process efficiency, customer trust, and organisational success.

Keywords

Quality Management Standards, Quality Management System (QMS), ISO 9001, Cost of Quality (CoQ), Prevention Cost, Appraisal Cost, Internal Failure Cost, External Failure Cost, Software Quality Management, Software Business Models.

Discussion

2.1.1 Quality Management Standards

Quality Management Standards are formally documented sets of principles, requirements, and best practices developed by recognised international organisations to help businesses establish effective quality management systems. These standards provide organisations with a common framework for planning, controlling, monitoring, and improving their processes to ensure consistent quality in products and services.

Simply producing a good product once is not sufficient for business success. Customers expect the same level of quality every time they purchase a product or use a service. Quality Management Standards help organisations achieve this consistency by standardizing their processes and ensuring that every activity follows predefined procedures.

These standards focus not only on the quality of the final product but also on the effectiveness of the processes used to create it. They encourage organisations to identify customer requirements, establish quality objectives, monitor performance, evaluate risks, and continuously improve their operations. As a result, organisations become more efficient, reduce waste, minimise errors, and improve customer satisfaction.

Another important aspect of Quality Management Standards is their emphasis on documentation. Organisations are required to document their policies, procedures, work instructions, and quality objectives. Proper documentation ensures that every employee performs tasks consistently and that organisational knowledge is preserved even when personnel change.

Quality Management Standards are applicable to organisations of all sizes, including manufacturing industries, software companies, healthcare institutions, educational organisations, financial institutions, government agencies, and service providers. Whether an organisation manufactures automobiles, develops software applications, provides banking services, or operates a hospital, these standards can be adapted to improve quality and operational efficiency.

For example, consider a software development company that develops mobile applications. Without standardised development procedures, different teams may

SGOU

follow different coding practices, testing methods, and documentation styles, resulting in inconsistent software quality. By implementing a Quality Management Standard such as ISO 9001, the company establishes standardized development processes, conducts regular quality reviews, maintains proper documentation, and continually improves its software development practices. As a result, customers receive more reliable software products with fewer defects and improved performance.

In simple terms, Quality Management Standards help organisations "do the right things in the right way every time." Rather than relying on chance or individual effort, they create a systematic approach that ensures quality becomes an integral part of every organisational activity. Quality Management Standards are documented guidelines that help organisations consistently produce high-quality products and services by following standardised processes and continuously improving their performance.

2.1.2 Objectives of Quality Management Standards

The following are the major objectives of Quality Management Standards.

1. To Achieve Customer Satisfaction

The most important objective of any Quality Management Standard is to ensure customer satisfaction. Customers are the foundation of every business, and their satisfaction determines the organization's success.

Customers expect products and services that meet their requirements in terms of quality, reliability, safety, performance, and timely delivery. When organisations consistently satisfy these expectations, customers develop trust and remain loyal to the organisation.

Quality Management Standards encourage organisations to identify customer requirements, monitor customer feedback, resolve complaints effectively, and continually improve products and services based on customer expectations.

2. To Ensure Consistent Quality

Consistency is one of the most important characteristics of quality. Customers expect the same level of quality every time they purchase a product or use a service.

Quality Management Standards help organisations establish standardised procedures for every activity. As a result, employees perform their tasks using the same methods, ensuring uniform quality regardless of who performs the work.

Consistency not only improves customer confidence but also strengthens the organisation's reputation in the market.

3. To Reduce Errors and Defects

One of the primary objectives of Quality Management Standards is to minimise errors during production and service delivery.

Instead of identifying defective products after production, these standards focus on preventing defects by improving process control, employee training, equipment

SGOU

maintenance, and quality monitoring.

Reducing defects helps organisations save time, reduce costs, and improve customer satisfaction.

4. To Improve Process Management

Every organisation performs its work through a series of interconnected processes. The quality of the final product depends on the effectiveness of each individual process.

Quality Management Standards encourage organisations to analyse every process, identify weaknesses, eliminate unnecessary activities, and continuously improve operational performance.

Efficient process management leads to higher productivity and better resource utilisation.

5. To Improve Decision Making

Effective management decisions should be based on accurate data rather than assumptions or personal opinions.

Quality Management Standards encourage organisations to collect performance data, analyse trends, evaluate customer feedback, monitor quality indicators, and make evidence-based decisions.

Data-driven decision making improves organisational planning and reduces business risks.

6. To Build Customer Trust and Organisational Reputation

Organisations that consistently deliver high-quality products earn the trust of customers and establish a strong reputation in the marketplace.

International quality certifications such as **ISO 9001** demonstrate that an organisation follows globally accepted quality management practices. This increases customer confidence and enhances business opportunities.

A strong reputation also helps organisations attract new customers, business partners, and investors.

2.1.3 Major Quality Management Standards

Quality Management Standards are internationally recognised frameworks that help organisations establish effective management systems for ensuring quality, improving efficiency, and meeting customer expectations. Different industries face different challenges and regulatory requirements. Therefore, several quality management standards have been developed to address the specific needs of various sectors such as manufacturing, healthcare, information technology, automotive, food processing, and environmental management.

Although each standard has a different area of focus, they all share a common objective: to help organisations consistently deliver high-quality products and services while

SGOU

continuously improving their business processes. These standards provide organisations with a systematic approach to planning, implementing, monitoring, evaluating, and improving their operations.

The following are some of the most widely adopted Quality Management Standards.

1. International Organisation for Standardisation 9001 – Quality Management System (QMS)

ISO 9001 is the world's most popular and widely implemented Quality Management Standard. It specifies the requirements for establishing, implementing, maintaining, and continually improving a Quality Management System (QMS). The standard was developed by the International Organisation for Standardisation (ISO) and is applicable to organisations of every size and type, including manufacturing industries, educational institutions, hospitals, software companies, banks, government organisations, and service providers.

The primary purpose of ISO 9001 is to ensure that organisations consistently provide products and services that meet customer expectations and comply with applicable legal and regulatory requirements. Rather than focusing only on the final product, ISO 9001 emphasises improving the processes that produce the product or service.

For example, a software company certified under ISO 9001 follows standardised procedures for software design, coding, testing, documentation, and customer support. As a result, software defects are reduced, project delivery becomes more reliable, and customer satisfaction improves.

2. International Organisation for Standardisation 14001 – Environmental Management System (EMS)

As industries continue to expand, protecting the environment has become a major global concern. Organisations are expected not only to produce quality products but also to minimise their impact on the environment. ISO 14001 provides a framework for establishing an Environmental Management System (EMS) that helps organisations manage their environmental responsibilities effectively.

ISO 14001 encourages organisations to identify environmental risks, reduce pollution, conserve natural resources, minimise waste generation, and comply with environmental laws and regulations.

For instance, a manufacturing company may implement ISO 14001 by reducing water consumption, recycling production waste, and controlling air emissions. These practices not only protect the environment but also reduce operational costs.

3. International Organisation for Standardisation 45001 – Occupational Health and Safety Management System (OHSMS)

Employees are one of the most valuable assets of any organisation. Providing a safe and healthy workplace is essential for improving employee well-being and organisational productivity. ISO 45001 specifies the requirements for establishing an Occupational Health and Safety Management System (OHSMS).

SGOU

The standard helps organisations identify workplace hazards, assess occupational risks, prevent accidents, and create a safe working environment for employees.

For example, a construction company implementing ISO 45001 ensures that workers wear protective equipment, receive safety training, and follow standardized safety procedures before entering hazardous work areas.

4. International Organisation for Standardisation 27001 – Information Security Management System (ISMS)

In the digital age, organisations store vast amounts of sensitive information, including customer records, financial transactions, business strategies, and employee data. Protecting this information from cyber threats has become a critical business requirement.

ISO 27001 provides a framework for establishing an Information Security Management System (ISMS) that helps organisations manage information security risks effectively.

The standard focuses on maintaining the Confidentiality, Integrity, and Availability (CIA) of information.

For example, an online banking system implementing ISO 27001 uses encryption, strong authentication, firewalls, regular security audits, and access control mechanisms to protect customer information.

5. International Automotive Task Force 16949 – Automotive Quality Management System

The automotive industry requires extremely high levels of quality because even a small manufacturing defect can lead to serious safety risks. IATF 16949 is an internationally recognised quality management standard specifically developed for organisations involved in the design, production, installation, and servicing of automotive products. The standard builds upon ISO 9001 by adding additional automotive-specific requirements.

For example, automobile manufacturers and component suppliers use IATF 16949 to improve manufacturing quality, reduce defects, strengthen supplier management, and enhance vehicle safety.

6. International Organisation for Standardisation 13485 – Medical Devices Quality Management System

Medical devices such as surgical instruments, diagnostic equipment, ventilators, and patient monitoring systems directly affect human health. Therefore, their design, manufacture, and maintenance must follow strict quality requirements.

ISO 13485 specifies the requirements for a Quality Management System specifically designed for organisations involved in manufacturing medical devices. The standard ensures that medical devices consistently meet customer expectations and regulatory requirements while maintaining patient safety.

For example, a company manufacturing blood glucose monitoring devices follows ISO 13485 to ensure accurate measurements, reliable performance, and compliance with healthcare regulations.

SGOU

2.1.3.1 Comparison of Major Quality Management Standards

Standard	Primary Focus	Typical Industries	Main Benefit
ISO 9001	Overall Quality Management	All industries	Customer satisfaction and continual improvement
ISO 14001	Environmental Management	Manufacturing, Energy, Construction	Environmental protection and sustainability
ISO 45001	Occupational Health & Safety	Construction, Manufacturing, Mining	Safe and healthy workplace
ISO 27001	Information Security	IT, Banking, Healthcare, Government	Protection of organisational information
IATF 16949	Automotive Quality	Automobile manufacturers and suppliers	Defect prevention and improved vehicle quality
ISO 13485	Medical Device Quality	Medical device manufacturers	Safe and reliable medical devices

2.1.4 Cost of Quality (CoQ)

In today's competitive software industry, delivering high-quality software is essential for customer satisfaction, business success, and long-term sustainability. Developing quality software, however, is not free. Organisations must invest time, money, human resources, and technology to ensure that software products meet customer expectations and perform reliably. These investments are collectively referred to as the Cost of Quality (CoQ).

Many people assume that quality simply increases the overall cost of software development. In reality, investing in quality often reduces the total cost of a project by preventing defects, minimising rework, reducing maintenance expenses, and avoiding customer complaints. Organisations that neglect quality may initially spend less during development but often incur much higher costs later due to software failures, bug fixes, warranty support, customer dissatisfaction, and damage to their reputation.

The concept of Cost of Quality helps organisations understand how much they spend to achieve quality and how much they lose because of poor quality. By analysing these costs, managers can identify areas where improvements are needed and make informed decisions about investing in quality assurance activities.

For example, consider a software company developing an online banking application. If the company spends additional time on code reviews, software testing, and security audits before releasing the application, the development cost may increase slightly. However, this investment can prevent serious security vulnerabilities and software failures after deployment. On the other hand, if the company skips these quality activities,

SGOU

it may later spend much more fixing defects, compensating customers, restoring data, and repairing its reputation. Thus, **spending on quality is an investment rather than an unnecessary expense.**

Cost of Quality (CoQ) refers to the total cost incurred by an organisation to ensure that a product or service meets the required quality standards, along with the costs resulting from failures when quality requirements are not met**.**

2.1.4.1 Objectives of Cost of Quality

The Cost of Quality concept aims to achieve the following objectives:

- To improve the overall quality of software products.
- To identify the costs associated with quality-related activities.
- To reduce the occurrence of defects.
- To minimise the cost of software failures.
- To improve customer satisfaction.
- To encourage continuous process improvement.
- To support better management decisions.
- To increase organisational profitability.

2.1.4.2 Components of Cost of Quality

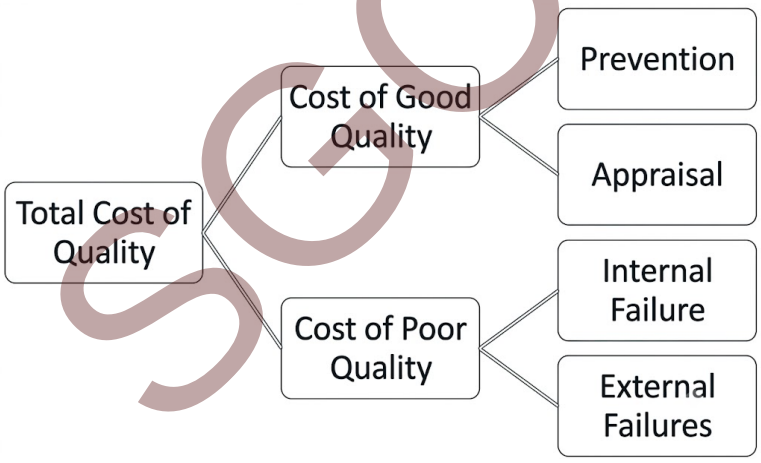


Fig 2.1.1 Components of Cost of Quality

The Cost of Quality (COQ) represents the total expenditure associated with achieving the required level of quality and dealing with the consequences of poor quality. It is broadly classified into two groups: Cost of Good Quality and Cost of Poor Quality. Cost of Good Quality refers to the costs incurred to ensure that products or services meet the required quality standards. These costs are mainly associated with preventing defects and identifying defects before they cause serious problems.

1. Prevention Cost

Prevention Costs are the costs incurred to avoid defects before they occur. These activities focus on improving processes, training employees, adopting standards,

SGOU

and implementing quality assurance practices so that errors are prevented rather than corrected later.

The basic principle behind prevention costs is that it is always cheaper to prevent a defect than to fix it after the software has been delivered.

Organisations that invest more in prevention generally experience fewer defects, lower maintenance costs, and higher customer satisfaction.

Before developing an e-commerce website, the development team conducts detailed requirement analysis, establishes coding guidelines, and provides secure coding training to developers. These activities help prevent defects and security vulnerabilities during development.

2. Appraisal Cost

Even with effective prevention activities, some defects may still occur during software development. Therefore, organisations perform various evaluation activities to identify defects before the software reaches customers. These costs are known as Appraisal Costs.

Appraisal Costs are incurred for measuring, evaluating, inspecting, and testing software products to verify that they meet specified quality requirements.

Unlike prevention activities, which attempt to stop defects from occurring, appraisal activities focus on detecting defects before product delivery.

A software company performs extensive functional testing, performance testing, and security testing before releasing a mobile banking application. The expenses incurred for these testing activities are appraisal costs.

Cost of Poor Quality refers to the costs that arise because a product or service does not meet the required quality standards. These costs are associated with failures and defects. Cost of Poor Quality consists of:

1. Internal Failure Costs

Despite prevention and appraisal activities, some defects may still be discovered before the software is delivered to customers. The expenses associated with correcting these defects are called Internal Failure Costs.

These costs occur inside the organization before product delivery.

Although internal failures increase development costs, they are generally less expensive than defects discovered by customers after product release.

2. External Failure Costs

External Failure Costs arise when software defects are discovered after the product has been delivered to customers. These are the most expensive quality-related costs because they directly affect customers and the organisation's reputation.

SGOU

External failures often result in customer complaints, software patches, technical support, warranty claims, legal liabilities, and even loss of future business.

For software organisations, external failures may also lead to cybersecurity incidents, financial losses, and damage to customer trust.

2.1.5 Business Models in Software Quality Management

In software engineering, business models play a significant role because they influence software development strategies, quality assurance practices, customer support, pricing, and product maintenance. A well-designed business model helps organisations identify their target customers, understand market demands, optimise operational processes, and deliver high-quality software solutions.

Different software companies follow different business models depending on the type of software they develop, the services they provide, and the needs of their customers. For example, a company developing customised software for a single client follows a different business model from a company selling commercial software to millions of users. Similarly, cloud-based software providers, open-source software organisations, and Software-as-a-Service (SaaS) companies each adopt unique business models suited to their business objectives.

A successful business model not only focuses on generating profit but also emphasises customer satisfaction, continuous innovation, quality improvement, and sustainable growth. Therefore, understanding different software business models is essential for software engineers, project managers, and entrepreneurs involved in software development.

A Business Model is a strategic framework that describes how an organisation creates value for its customers, delivers products or services, and generates revenue while maintaining profitability and achieving its business objectives.

2.1.5.1Types of Software Business Models

1. Product-Based Business Model

In a Product-Based Business Model, the organisation develops a standardised software product and sells it to multiple customers. The same software product is used by many users with little or no customisation.

2. Service-Based Business Model

Organisations develop customised software according to individual customer requirements

3. Software as a Service (SaaS)

Software is hosted on cloud servers and accessed through the Internet. Customers pay monthly or yearly subscription fees.

SGOU

4. Open-Source Business Model

The software source code is freely available for modification and distribution.

5. Freemium Business Model

The basic version of the software is provided free of charge, while premium features require payment.

6. Subscription Business Model

Customers pay a recurring monthly or annual fee to use the software.

A Business Model is the foundation of every software organisation. It defines how a company creates value, develops and delivers software, generates revenue, and sustains long-term growth. In Software Quality Management, business models play a crucial role because they influence quality assurance practices, customer support, software maintenance, and continuous improvement. Different business models—such as Product-Based, Service-Based, SaaS, Open-Source, Freemium, and Subscription—adopt different approaches to software development and quality management. A well-designed business model not only improves profitability but also ensures that software products remain reliable, secure, maintainable, and capable of satisfying customer needs. Understanding these models helps software professionals select appropriate strategies for delivering high-quality software while achieving sustainable business success.

Recap

- Software Quality Management (SQM) is the process of planning, assuring, controlling, and improving the quality of software throughout its development life cycle.
- The primary goal of Software Quality Management is to develop reliable, efficient, secure, maintainable, and customer-satisfying software.
- Quality Planning identifies quality objectives, standards, procedures, and resources required to achieve software quality.
- Quality Assurance (QA) focuses on preventing defects by ensuring that software development processes follow established standards and best practices.
- Quality Control (QC) focuses on detecting and correcting defects through inspections, reviews, and software testing.
- Continuous Quality Improvement (CQI) encourages organisations to regularly evaluate and improve software processes, products, and services.

SGOU

- Software Quality Standards provide internationally accepted guidelines for developing, evaluating, and maintaining high-quality software.
- The ISO 9000 Family establishes Quality Management System (QMS) requirements that help organisations improve process consistency and customer satisfaction.
- ISO/IEC 9126 defines six software quality characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability.
- ISO/IEC 25010 is the updated software quality model that defines eight characteristics, including Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability.
- Capability Maturity Model (CMM) evaluates the maturity of software development processes through five maturity levels, from Initial to Optimising.
- Capability Maturity Model Integration (CMMI) improves organisational processes by integrating multiple process improvement practices into a single framework.
- ISO/IEC 12207 defines standardised software life cycle processes, including primary, supporting, and organisational processes.

Objective Type Questions

1. What is the world's most widely implemented Quality Management Standard?
2. Name the standard used for Environmental Management.
3. Identify the standard for Occupational Health and Safety Management.
4. State the standard used for Information Security Management.
5. Name the Quality Management Standard for medical devices.
6. Expand the abbreviation CoQ.
7. What type of Cost of Quality prevents defects before they occur?
8. Name the Cost of Quality associated with software testing.
9. Identify the Cost of Quality incurred before product delivery.
10. What type of Cost of Quality occurs after product delivery?

SGOU

11. Name the cloud-based software business model.
12. Identify the business model in which source code is freely available.
13. What business model provides basic features free of cost?
14. Name the Quality Management Standard used in the automotive industry.

Answers to Objective Type Questions

1. ISO 9001.
2. ISO 14001.
3. ISO 45001
4. ISO/IEC 27001.
5. ISO 13485.
6. Cost of Quality (CoQ).
7. Prevention Cost.
8. Appraisal Cost.
9. Internal Failure Cost.
10. External Failure Cost.
11. Software as a Service (SaaS).
12. Open Source Software (OSS).
13. Freemium Model.
14. IATF 16949 (International Automotive Task Force 16949).

SGOU

Assignments

1. Describe the major Quality Management Standards.

2. Explain the different software business models used in Software Quality Management.

3. A software company plans to develop an online banking application. Recommend suitable Quality Management Standards for the organisation and explain how the implementation of these standards, along with effective Cost of Quality practices, can improve software quality, security, customer satisfaction, and business performance.

4. Discuss the four components of CoQ.

Suggested Reading

1. Evans, J. R., & Lindsay, W. M. (2023). *Managing for quality and performance excellence* (12th ed.). Cengage Learning.

2. Goetsch, D. L., & Davis, S. B. (2021). *Quality management for organizational excellence: Introduction to total quality* (9th ed.). Pearson.

3. Sommerville, I. (2021). *Software engineering* (11th ed.). Pearson.

4. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

5. Galin, D. (2018). *Software quality: Concepts and practice* (2nd ed.). Wiley.

Reference

1. International Organization for Standardization. (2022). *Information security, cybersecurity and privacy protection—Information security management systems—Requirements (ISO Standard No. 27001:2022).* International Organization for Standardization.

2. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.

3. Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

4. International Organization for Standardization. (2015). *Environmental management systems—Requirements with guidance for use (ISO Standard No. 14001:2015).* International Organization for Standardization.

SGOU



Review Techniques and Quality Frameworks

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept and importance of software reviews
- describe Personal Review and Desk-Check Review with suitable examples
- apply a review checklist to examine a software work product
- analyse the differences between Personal Review and Desk-Check Review
- analyse software quality standards and quality models based on their major characteristics

Prerequisite

Consider a software developer who has completed a program for an online shopping application. Before submitting the program for testing, the developer needs to check whether the requirements are correctly implemented, the calculations are accurate, the input is properly validated, and the coding standards are followed. This basic checking helps identify simple errors before they become difficult to correct. Therefore, learners should have a basic understanding of software engineering, the software development life cycle, software requirements, software testing, and software quality before studying this unit. Familiarity with source code, software documentation, and basic software quality assurance activities will also be helpful.

Studying review techniques and quality frameworks helps learners understand how software quality can be improved throughout the development process. Knowledge of Personal Review and Desk-Check Review enables learners to identify defects at an early stage, reduce rework, and improve the consistency and reliability of software work products.

Understanding quality standards and software quality models also helps learners evaluate software using systematic quality characteristics. These concepts provide a foundation for developing and evaluating software that is reliable, usable, secure, maintainable, and suitable for its intended purpose.

SGOU

Keywords

Software Review, Personal Review, Desk-Check Review, Review Checklist, Software Quality, Quality Standard, Quality Model, McCall's Quality Model, Boehm's Quality Model, ISO/IEC 25010

Discussion

2.2.1 Introduction

Software quality cannot be achieved only by testing the completed software product. Quality must be considered throughout the software development process. Errors introduced during requirements analysis, system design, coding, documentation, and other development activities can increase development cost and affect the reliability of the final software product. Therefore, organisations use different software review techniques to identify errors at an early stage and improve the quality of software work products.

A software review is a systematic examination of a software work product such as a requirements document, design, source code, test plan, database design, or user manual. Reviews help identify errors, omissions, inconsistencies, and deviations from requirements or standards before they become more difficult and expensive to correct.

This unit focuses mainly on Personal Review and Desk-Check Review. Personal Review involves examining one's own work carefully before submitting it to others. Desk-Check Review involves systematically checking a work product against requirements, specifications, standards, rules, or checklists. In addition to review techniques, the unit introduces quality standards and software quality models that help organisations define, evaluate, and improve software quality.

2.2.2 Relevance of Review Techniques and Quality Frameworks

Consider a software development team developing an online shopping application. During development, a programmer may accidentally use an incorrect variable name, omit an important validation condition, or write code that does not follow the organisation's coding standards. If such errors are discovered only during final testing, correcting them may require additional development time and effort. A Personal Review allows the programmer to examine the code before submitting it, while a Desk-Check Review can be used to systematically verify the code against requirements, coding standards, and a checklist. Similarly, quality frameworks and models help the organisation establish a systematic approach to software quality. Therefore, review techniques and quality frameworks are relevant because they help identify problems early, reduce rework, improve consistency, support continuous improvement, and increase the reliability of software products.

SGOU

2.2.3 Software Review

A Software Review is a systematic examination of a software work product to identify errors, omissions, inconsistencies, and opportunities for improvement. A work product may be a requirements document, design document, source code, test plan, user manual, database design, or other software-related document. Software reviews are an important part of software quality assurance because they help identify defects before the software reaches the testing or production stage.

For example, suppose a software developer creates a login module for a banking application. The developer may review the code to check whether the username and password are validated correctly, whether invalid input is handled properly, and whether security requirements are followed. Identifying these issues during review is much easier and less expensive than discovering them after deployment.

2.2.3.1 Objectives of Software Review

The major objectives of software review are:

1. **To identify defects early**
   Reviews help identify errors before they reach later stages of development.

2. **To improve software quality**
   Reviewing software work products helps improve correctness, consistency, reliability, and maintainability.

3. **To verify requirements**
   Reviews help determine whether the software work product satisfies specified requirements.

4. **To ensure compliance with standards**
   Reviews verify whether developers follow organisational standards, coding guidelines, and documentation requirements.

5. **To reduce rework**
   Early identification of defects reduces the amount of work required to correct them later.

6. **To improve team communication**
   Group-based reviews allow developers to share knowledge and understand project requirements.

7. **To support continuous improvement**
   Review findings can be used to improve development processes and prevent similar defects in future projects.

SGOU

2.2.3.2 Types of Software Reviews

Software reviews can be performed using different techniques depending on the nature of the work product and the purpose of the review.

Common review techniques include:

- Personal Review
- Desk-Check Review
- Walkthrough
- Inspection
- Peer Review
- Formal Technical Review

In this unit, special attention is given to Personal Review and Desk-Check Review.

a. Personal Review

A **Personal Review** is a review performed by an individual on their own work before the work product is submitted, shared, tested, or used by others.

It is one of the simplest forms of software review. The developer or author carefully examines the work to identify errors, omissions, inconsistencies, and violations of requirements or standards.

Personal Review is sometimes considered an important first level of quality checking because the person who creates a work product is responsible for checking its correctness before asking others to review it.

Example

Consider a programmer who has developed a Python program to calculate the total amount of an online shopping order. Before submitting the program for testing, the programmer checks:

- Whether all variables are correctly declared.
- Whether the calculation is correct.
- Whether input validation is included.
- Whether appropriate error handling is provided.
- Whether coding standards are followed.
- Whether unnecessary code is present.

If the programmer discovers that the discount calculation is incorrect during this review, the error can be corrected before formal testing.

SGOU

Steps in Personal Review

1. Understand the requirements.
2. Examine the completed work.
3. Check the logic and calculations.
4. Check for missing information.
5. Compare the work with relevant standards.
6. Identify and record errors.
7. Correct the identified errors.
8. Perform a final check.

Advantages of Personal Review

- Simple and inexpensive.
- Can be performed at any stage.
- Helps identify defects early.
- Reduces rework.
- Encourages individual responsibility for quality.
- Does not require other team members.
- Encourages compliance with coding and documentation standards.

Limitations of Personal Review

- The author may fail to notice their own mistakes.
- Personal assumptions may influence the review.
- Complex defects may remain undetected.
- It does not provide multiple viewpoints.
- Its effectiveness depends on the knowledge and experience of the developer.

Therefore, Personal Review is useful as an initial quality check, but it may need to be supported by other review techniques.

b. Desk-Check Review

A Desk-Check Review is a systematic manual examination of a software work product by checking it against requirements, specifications, standards, rules, or predefined checklists.

SGOU

The term "desk-check" refers to checking the work at the developer's or reviewer's desk before the software proceeds to formal testing or execution. It is particularly useful for checking source code, algorithms, logic, calculations, requirements, and documentation.

Unlike simply reading the work casually, a Desk-Check Review follows a more organised approach.

Example

Suppose a developer writes a program to calculate the average marks of five learners. During a Desk-Check Review, the developer manually follows the logic using sample values.

Suppose the marks are:

60, 70, 80, 90, 100

The reviewer checks whether:

- All five values are accepted.
- The values are stored correctly.
- The total is calculated correctly.
- The number of values is correctly identified as five.
- The average is calculated correctly.
- Invalid values are handled appropriately.
- The final result is displayed correctly.

This process can reveal logical errors before the program is executed.

Desk-Check Review Procedure

The following procedure can be used:

Step 1: Identify the Work Product

Identify the source code, algorithm, requirement, document, or design to be reviewed.

Step 2: Understand the Requirements

Study the requirements or specifications.

Step 3: Prepare a Checklist

Prepare a list of items that need to be verified.

Step 4: Examine the Work

Check each part systematically.

SGOU

Step 5: Trace the Logic

For programs and algorithms, manually follow the execution using sample inputs.

Step 6: Record Defects

Record all errors, omissions, and inconsistencies.

Step 7: Correct the Defects

The developer corrects the identified problems.

Step 8: Perform Follow-up

Review the corrected work again.

Advantages of Desk-Check Review

Desk-Check Review provides the following advantages:

- Helps detect logical errors before execution.
- Provides a systematic method of checking software work products.
- Can identify missing requirements.
- Helps verify compliance with standards.
- Reduces the number of defects entering formal testing.
- Requires relatively few resources.
- Can be performed before software execution.
- Supports better documentation and coding practices.
- Helps improve software reliability.

Limitations of Desk-Check Review

- Some limitations of Desk-Check Review are:
- It may be time-consuming for large programs.
- It depends on the skill of the reviewer.
- Complex interactions may be difficult to identify manually.
- It cannot replace actual software testing.
- Reviewers may overlook defects if the checklist is incomplete.
- It may become ineffective if performed without sufficient attention.

Therefore, Desk-Check Review should be considered a complementary quality activity rather than a replacement for software testing.

SGOU

2.2.3.4 Personal Review and Desk-Check Review

Both techniques are useful, but their approaches are different.

Table 2.2.1: Comparison of Personal Review and Desk-Check Review

Feature	Personal Review	Desk-Check Review
Reviewer	Author/developer	Developer or reviewer
Main purpose	Check one's own work	Systematically examine work
Approach	Individual	Structured and systematic
Checklist	Optional	Usually recommended
Requirements checking	Basic	Systematic
Logic tracing	May be performed	Commonly performed
Multiple viewpoints	No	May be possible
Cost	Very low	Low
Best use	Initial quality check	Detailed manual verification

2.2.4 Quality Standards

A Quality Standard is a documented set of requirements, guidelines, principles, or best practices used to ensure that products, services, or processes achieve specified quality objectives.

Quality standards provide organisations with a common framework for managing and improving quality.

For example, ISO 9001 provides requirements for establishing a Quality Management System. An organisation can use these requirements to define processes, monitor performance, manage risks, evaluate customer satisfaction, and continuously improve its operations.

Quality standards are important because they help organisations achieve:

- Consistent quality
- Process control
- Customer satisfaction
- Risk reduction
- Continuous improvement

SGOU

- Regulatory and contractual compliance
- Better organisational performance

2.2.5 Software Quality Models

A Software Quality Model is a structured framework used to identify and evaluate the characteristics of software quality.

A quality model helps answer the question:

What characteristics should a high-quality software product possess?

Software quality is not limited to whether a program produces the correct output. A good software product should also be reliable, secure, usable, maintainable, efficient, and compatible with its environment.

Three important software quality models are:

1. McCall's Quality Model
2. Boehm's Quality Model
3. ISO/IEC 25010 Quality Model

- McCall's Quality Model

McCall's Quality Model is one of the early software quality models. It organises software quality factors into three broad categories:

1. Product Operation

This category focuses on how the software operates.

The important factors are:

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability

2. Product Revision

This category focuses on how easily software can be modified.

The important factors are:

- Maintainability
- Flexibility
- Testability

SGOU

3. Product Transition

This category focuses on the ability of software to adapt to different environments.

The important factors are:

- Portability
- Reusability
- Interoperability

2. Boehm's Quality Model

Boehm's Quality Model is a hierarchical software quality model that evaluates the quality of software through different levels of characteristics. At the highest level, it identifies three major characteristics: As-is Utility, Maintainability, and Portability. As-is Utility describes how effectively the software performs its intended functions in the current environment and includes characteristics such as reliability, efficiency, and human engineering. Maintainability refers to the ease of understanding, testing, correcting, and modifying the software, while Portability refers to the ability of software to operate in different environments with minimal changes.

The model further divides these characteristics into more specific attributes such as testability, understandability, modifiability, accuracy, completeness, consistency, and device independence. For example, an online banking application should provide accurate transactions, reliable performance, an easy-to-use interface, and strong maintainability so that new features and security updates can be added easily. Thus, Boehm's model helps organisations evaluate software quality systematically by considering not only whether the software works correctly, but also whether it is easy to use, maintain, modify, and adapt to different environments.

3. ISO/IEC 25010 Software Product Quality Model

ISO/IEC 25010 provides a widely used model for evaluating software product quality. The model identifies important quality characteristics that can be considered when designing, developing, testing, and evaluating software.

The major software product quality characteristics include:

1. Functional Suitability

Functional suitability refers to the extent to which software provides functions that meet stated and implied needs.

For example, an online banking application should correctly provide functions such as account balance checking, fund transfer, bill payment, and transaction history.

2. Performance Efficiency

Performance efficiency refers to how effectively software performs in relation to the resources used.

SGOU

For example, a banking application should respond quickly without unnecessarily consuming memory, processing power, or network resources.

3. Compatibility

Compatibility refers to the ability of software to work effectively with other systems, products, or environments.

For example, a web application should operate correctly with different browsers and integrate with required external services.

4. Usability

Usability refers to how easily users can learn, understand, and operate the software.

For example, an online shopping application should provide simple navigation and clear instructions.

5. Reliability

Reliability refers to the ability of software to perform consistently under specified conditions for a specified period.

For example, an online examination system should remain available and function correctly during an examination.

6. Security

Security refers to the ability of software to protect information and prevent unauthorized access.

For example, banking software should protect customer accounts through authentication, authorisation, encryption, and other security mechanisms.

7. Maintainability

Maintainability refers to how easily software can be analysed, modified, corrected, and improved.

Well-structured and properly documented software is generally easier to maintain.

8. Portability

Portability refers to the ability of software to be transferred from one environment to another.

For example, software that can operate on different operating systems with minimal modification has better portability.

SGOU

2.2.6 Quality Standards and Quality Models

Quality standards and quality models are related, but they serve different purposes.

Table 2.2.2: Comparison of Quality Standards and Quality Models

Aspect	Quality Standard	Quality Model
Meaning	Defines requirements or guidelines	Defines quality characteristics
Main question	How should quality be managed?	What characteristics represent quality?
Main focus	Processes and requirements	Software or product characteristics
Example	ISO 9001	ISO/IEC 25010
Main use	Establish quality practices	Evaluate software quality
Result	Consistent quality processes	Structured quality assessment

Recap

- Software quality must be considered throughout the software development process and not only during final testing.
- Software Review is a systematic examination of a software work product to identify errors, omissions, inconsistencies, and areas for improvement.
- Software reviews can be performed on requirements documents, design documents, source code, test plans, database designs, and user manuals.
- The main purpose of software review is to identify defects early and reduce the effort and cost required for correcting them later.
- Software reviews also help in verifying requirements, following standards, improving quality, reducing rework, and supporting continuous improvement.
- Personal Review is the process of examining one's own work before submitting it, testing it, or making it available to others.
- Personal Review is simple, inexpensive, and useful for early defect detection, but the author may fail to identify their own mistakes.
- Desk-Check Review is a systematic manual examination of a software work product against requirements, specifications, standards, rules, or checklists.

SGOU

- Desk-Check Review is particularly useful for checking source code, algorithms, program logic, calculations, requirements, and documentation.
- A Desk-Check Review generally involves understanding requirements, preparing a checklist, examining the work, tracing logic, recording defects, correcting defects, and performing follow-up.
- Personal Review and Desk-Check Review are complementary techniques. Personal Review focuses mainly on the author's own work, while Desk-Check Review provides a more systematic examination.
- A Quality Standard provides documented requirements, guidelines, or principles that help organisations manage and improve the quality of their products, services, and processes.
- A Software Quality Model provides a structured framework for identifying and evaluating important characteristics of software quality.
- McCall's Quality Model, Boehm's Quality Model, and ISO/IEC 25010 are important approaches for understanding and evaluating software quality.
- Review techniques, quality standards, and quality models work together to support early defect detection, consistent processes, systematic quality evaluation, continuous improvement, and the development of reliable software products.

Objective Type Questions

1. What is the systematic examination of a software work product to identify errors and inconsistencies called?
2. Which software review technique involves examining one's own work before submitting it to others?
3. Which review technique involves systematically checking a work product against requirements, standards, or checklists?
4. What is the main purpose of identifying defects early in the software development process?
5. Name the review technique that is considered an initial quality check performed by the author or developer.
6. What is a documented set of requirements, guidelines, or principles used to achieve specified quality objectives called?

SGOU

7. Which quality model organises software quality factors into Product Operation, Product Revision, and Product Transition?

8. Which software quality model focuses on characteristics such as reliability, efficiency, usability, testability, understandability, modifiability, and portability?

9. Which ISO standard provides a structured model for evaluating software product quality?

10. Which software quality characteristic refers to the ability of software to perform consistently under specified conditions?

11. Which software quality characteristic refers to the ability of software to protect information and prevent unauthorised access?

12. Which software quality characteristic refers to the ability of software to be transferred from one environment to another?

Answers to Objective Type Questions

1. Software Review.
2. Personal Review.
3. Desk-Check Review.
4. To reduce the effort and cost of correcting defects later.
5. Personal Review.
6. Quality Standard.
7. McCall's Quality Model.
8. Boehm's Quality Model.
9. ISO/IEC 25010.
10. Reliability.
11. Security.
12. Portability.

SGOU

Assignments

1. Explain the importance of software review in software development.
2. Describe the procedure of Personal Review.
3. Describe the procedure of Desk-Check Review.
4. Differentiate between Personal Review and Desk-Check Review.
5. Describe the major characteristics of the ISO/IEC 25010 software quality model.
6. Analyse the differences between Personal Review and Desk-Check Review based on their approach and application.
7. Analyse the role of quality standards and software quality models in improving software quality.

Suggested Reading

1. Sethi, R. (2023). *Software engineering: Basic principles and best practices*. Cambridge University Press.
2. O'Regan, G. (2022). *Concise guide to software engineering: From fundamentals to application methods*. Springer.
3. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). *Essentials of software testing*. Cambridge University Press.
4. Jalote, P. (2019). *An integrated approach to software engineering* (4th ed.). Springer.
5. Mall, R. (2018). *Fundamentals of software engineering* (5th ed.). PHI Learning.

SGOU

Reference

1. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.

2. International Organization for Standardization & International Electrotechnical Commission. (2011). *Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models (ISO/IEC 25010:2011).* International Organization for Standardization.

3. International Organization for Standardization. (2015). *Quality management systems—Requirements (ISO 9001:2015).* International Organization for Standardization.

SGOU



Formal Review Methods

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept and purpose of formal software reviews.
- describe the procedure of a Walk-Through Review.
- apply a suitable checklist during a Walk-Through or Inspection Review.
- analyse the major characteristics of Walk-Through and Inspection Review.
- differentiate between Walk-Through Review and Inspection Review based on their procedures and objectives.

Prerequisite

Consider a software team that has completed the design of an online examination system. Before coding begins, the team needs to examine the design carefully to determine whether all requirements have been addressed and whether there are any missing or incorrect design decisions. A developer may present the design to other team members during a Walk-Through, allowing them to examine the work and identify possible problems. A more formal Inspection Review can then be conducted using defined roles, procedures, and checklists. Therefore, learners should have a basic understanding of software engineering, software development life cycle, software requirements, software quality, software reviews, Personal Review, and Desk-Check Review before studying this unit.

Studying formal review methods helps learners understand how software work products can be examined systematically before implementation or testing. Knowledge of Walk-Through and Inspection Review helps in identifying defects early, improving communication among team members, reducing rework, and ensuring that requirements and standards are followed. These methods also help learners understand how structured review activities contribute to the development of reliable and high-quality software.

SGOU

Keywords

Formal Review, Walk-Through Review, Inspection Review, Moderator, Reviewer, Reader, Recorder, Author, Review Checklist, Defect Detection, Rework, Follow-up, Software Quality

Discussion

2.3.1 Introduction

Software development involves several work products such as requirements documents, design documents, source code, test cases, and user documentation. Errors in these work products can affect the quality, reliability, and maintainability of the final software. Testing is useful for identifying failures in executable software, but many defects can be identified before the software is executed. Formal review methods provide a systematic approach for examining software work products and identifying defects at an early stage.

Two important formal review methods are Walk-Through Review and Inspection Review. A Walk-Through is a review in which the author or developer presents a work product to a group of reviewers and explains its content, logic, and decisions. The reviewers examine the work product and provide comments or identify possible defects. An Inspection Review is a more structured and formal review method that follows defined procedures, roles, checklists, and review activities. Both methods contribute to early defect detection and improved software quality.

Consider a development team preparing the source code for a college management system. The programmer may understand the program correctly, but another person may identify an error that the programmer has overlooked. For example, a condition may be missing from the student admission process or an important validation rule may not have been implemented. During a Walk-Through, the programmer can explain the code to other members of the team, who can examine the logic and identify possible problems. In an Inspection Review, the same work product can be examined using a more formal procedure and checklist.

Formal review methods are therefore relevant because they help identify defects before they become expensive to correct. They also improve communication, promote adherence to standards, encourage systematic quality checking, and reduce the number of defects that reach later stages of development.

2.3.2 Formal Review Methods

A Formal Review is a planned and systematic examination of a software work product performed according to a defined procedure.

SGOU

A work product may include:

- Requirements specification
- System design
- Source code
- Database design
- Test cases
- User documentation
- Project plans

Formal reviews involve defined objectives, preparation, examination, recording of findings, and follow-up activities.

The two formal review methods discussed in this unit are:

Walk-Through Review

Inspection Review

2.3.2.1Walk-Through Review

A Walk-Through Review is a review in which the author or developer presents a software work product to a group of reviewers and explains the work product step by step.

The main purpose is to examine the work, identify possible defects, obtain feedback, and improve understanding among team members.

The author generally takes an active role in the review and explains the logic, design decisions, requirements, or implementation details.

Objectives of Walk-Through Review

The major objectives of a Walk-Through Review are:

1. To identify defects in the work product.
2. To obtain feedback from other team members.
3. To improve understanding of the work product.
4. To verify whether requirements have been addressed.
5. To identify missing or inconsistent information.
6. To encourage knowledge sharing.
7. To improve the quality of the work product.

SGOU

Participants in a Walk-Through

A Walk-Through generally involves:

1. Author

The person who created the work product. The author normally presents and explains the work.

2. Reviewers

People who examine the work product and provide comments or identify possible defects.

3. Moderator or Coordinator

A person who may organise the review and ensure that the discussion remains focused.

The exact participants may vary depending on the organisation and type of work product.

Walk-Through Review Procedure

A typical Walk-Through can be performed through the following steps:

Step 1: Select the Work Product

The work product to be reviewed is identified.

Step 2: Define the Objective

The purpose of the review is clearly established.

Step 3: Distribute the Work Product

The work product is provided to the participants before the meeting whenever possible.

Step 4: Individual Preparation

Reviewers examine the work product and note possible questions or defects.

Step 5: Presentation

The author presents the work product and explains its important sections, logic, and decisions.

Step 6: Discussion

Reviewers ask questions and provide comments about the work product.

Step 7: Record Findings

Identified defects, questions, and observations are recorded.

Step 8: Correct the Defects

The author or responsible developer makes the necessary corrections.

SGOU

Step 9: Follow-up

The corrected work product is checked again when required.

Example of Walk-Through Review

Suppose a developer has prepared the design of an **online examination system**.

During the Walk-Through, the developer explains:

- User registration
- Login process
- Question selection
- Answer submission
- Automatic evaluation
- Result generation

The reviewers examine the design while the developer explains each part.

One reviewer may identify that the design does not specify what happens if the network connection is lost during answer submission. This issue can be recorded and addressed before implementation.

Advantages of Walk-Through Review

Walk-Through Review provides several advantages:

- Helps identify defects early.
- Encourages communication among team members.
- Provides multiple viewpoints.
- Helps clarify requirements and design decisions.
- Promotes knowledge sharing.
- Helps identify missing information.
- Can improve the quality of documentation and source code.
- Provides an opportunity for the author to receive feedback.

Limitations of Walk-Through Review

Some limitations of Walk-Through Review are:

- The effectiveness depends on the preparation of reviewers.
- The review may become too dependent on the author.

SGOU

- Important defects may be missed if the discussion is not systematic.
- Large work products may require considerable time.
- Reviewers may hesitate to criticise the author's work.
- Lack of preparation can reduce the effectiveness of the review.

Therefore, a Walk-Through is useful for collaborative examination and understanding, while more formal methods may be required when strict and systematic defect detection is needed.

2.3.2.2 Inspection Review

An Inspection Review is a highly structured and formal review method used to identify defects in software work products.

Inspection is more systematic than an informal review because it follows defined procedures, responsibilities, preparation activities, checklists, and recording mechanisms.

The main purpose of an Inspection Review is to identify defects rather than simply discuss the work product.

Inspection can be applied to:

- Requirements
- Design documents
- Source code
- Test plans
- Test cases
- Documentation

Objectives of Inspection Review

The major objectives of Inspection Review are:

- To identify defects systematically.
- To verify compliance with requirements.
- To verify compliance with standards.
- To reduce defects in later stages.
- To improve software quality.

SGOU

- To reduce rework.
- To collect information about common defects.
- To support process improvement.

Participants in Inspection Review

An Inspection Review normally involves clearly defined roles.

1. Author

The person who created the work product being inspected.

2. Moderator

The person responsible for planning and coordinating the inspection.

3. Reviewer

A person who examines the work product to identify defects.

4. Reader

The person who guides the inspection team through the work product.

5. Recorder

The person who records identified defects and observations.

The exact roles may vary between organisations, but defining responsibilities helps make the inspection systematic.

Inspection Review Procedure

A typical Inspection Review can be organised into the following stages:

Planning- Overview - Preparation - Inspection Meeting - Rework - Follow-up

Step 1: Planning

The moderator selects the work product and schedules the inspection.

Step 2: Overview

The author provides necessary background information about the work product.

Step 3: Individual Preparation

Reviewers study the work product independently and identify possible defects using suitable checklists.

Step 4: Inspection Meeting

The inspection team examines the work product systematically.

SGOU

Step 5: Recording Defects

All identified defects are documented.

Step 6: Rework

The author corrects the identified defects.

Step 7: Follow-up

The moderator or responsible person verifies whether the defects have been corrected properly.

Inspection Checklist

A checklist can help reviewers examine a work product systematically.

Table 2.3.1: Sample Inspection Checklist

Review Item	Yes/No	Remarks
All requirements are addressed		
Requirements are consistent		
Design follows requirements		
Coding standards are followed		
Input validation is provided		
Error handling is appropriate		
Security requirements are considered		
Documentation is complete		
Unnecessary code or information is removed		
Identified defects are recorded		

A checklist helps reviewers maintain consistency and reduces the possibility of overlooking important review items.

Example of Inspection Review

Consider the source code of a student attendance management system.

The inspection team examines the code using a checklist.

The reviewers identify the following defects:

- Attendance percentage is not calculated correctly.
- Invalid attendance values are accepted.

SGOU

- One required validation condition is missing.
- A coding standard has not been followed.
- Error handling is incomplete.

The defects are recorded during the inspection. The developer then corrects them, and a follow-up is conducted to verify the corrections.

This example shows how Inspection Review provides a structured approach to defect identification and correction.

Advantages of Inspection Review

Inspection Review provides several advantages:

- Provides a systematic review process.
- Helps detect defects early.
- Uses defined roles and responsibilities.
- Supports the use of checklists.
- Helps identify recurring types of defects.
- Reduces rework.
- Improves compliance with standards.
- Improves software quality.
- Supports process improvement.
- Reduces the possibility of defects reaching later stages.

Limitations of Inspection Review

Inspection Review also has some limitations:

- Requires careful planning and preparation.
- Can require more time than less formal review methods.
- Requires trained reviewers.
- May involve several team members.
- Can become difficult for very large work products.
- Requires proper recording and follow-up.
- Poorly conducted inspections may not provide the expected benefits.

SGOU

2.3.3 Walk-Through Review and Inspection Review

Walk-Through and Inspection Review are both systematic review methods, but they differ in their level of formality and procedure.

Table 2.3.2 : Comparison of Walk-Through and Inspection Review

Feature	Walk-Through Review	Inspection Review
Formality	Moderately formal	Highly formal
Main presenter	Usually the author	Reader or designated reviewer
Main purpose	Understanding, feedback, and defect identification	Systematic defect identification
Roles	Less strictly defined	Clearly defined
Checklist	May be used	Commonly used
Preparation	Required	More structured
Meeting	Presentation and discussion	Structured inspection
Recording	Findings are recorded	Defects are formally recorded
Follow-up	May be performed	Normally required
Main focus	Discussion and feedback	Defect detection

SGOU

Recap

- Formal reviews are systematic examinations of software work products.
- Formal reviews help identify defects before they reach later stages of development.
- Walk-Through Review involves presenting and explaining a work product to reviewers.
- The author generally plays an active role in a Walk-Through.
- Walk-Throughs support defect identification, communication, feedback, and knowledge sharing.
- Inspection Review is a highly structured method of examining software work products.
- Inspection uses defined procedures, roles, preparation, checklists, and follow-up activities.

- Important inspection roles include author, moderator, reviewer, reader, and recorder.
- A checklist helps reviewers examine work products systematically.
- Inspection findings should be recorded and communicated for correction.
- Rework involves correcting defects identified during the review.
- Follow-up verifies whether identified defects have been properly corrected.
- Walk-Through is generally less formal than Inspection Review.
- Reviews can identify defects without executing the software.
- Walk-Through and Inspection Review complement software testing and contribute to improved software quality.

Objective Type Questions

1. What is a formal review?
2. Which review method involves the author presenting a work product to reviewers?
3. Who normally presents the work product during a Walk-Through?
4. What is the main purpose of an Inspection Review?
5. Who coordinates an Inspection Review?
6. Who records defects identified during an inspection?
7. What is used to ensure systematic checking during an Inspection Review?
8. What activity involves correcting defects identified during a review?
9. What activity verifies whether identified defects have been corrected?
10. Which review method is more formal, Walk-Through or Inspection?

SGOU

Answers to Objective Type Questions

1. A systematic examination of a software work product.
2. Walk-Through Review.
3. The author or developer.
4. Systematic identification of defects.
5. The moderator.
6. The recorder.
7. A review checklist.
8. Rework.
9. Follow-up.
10. Inspection Review.

Assignments

1. Why is Inspection Review considered more formal than Walk-Through Review?
2. Explain why formal reviews cannot completely replace software testing.
3. How does a Walk-Through help in knowledge sharing?
4. Analyse the differences between Walk-Through Review and Inspection Review.
5. Analyse how Inspection Review helps in early defect detection and reduction of rework.

SGOU

Suggested Reading

1. Sethi, R. (2023). *Software engineering: Basic principles and best practices*. Cambridge University Press.

2. O'Regan, G. (2022). *Concise guide to software engineering: From fundamentals to application methods*. Springer.

3. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). *Essentials of software testing*. Cambridge University Press.

4. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.

5. Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

Reference

1. Jalote, P. (2025). *A concise introduction to software engineering: With open source and GenAI* (2nd ed.). Springer.

2. Wang, et al. (2024). Enhancing human-machine pair inspection with risk number and code inspection diagram. *Software Quality Journal, 32*, 939–959

3. Pressman, R. S., & Maxim, B. R. (2023). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.

4. Pfleeger, S. L., & Atlee, J. M. (2020). *Software engineering: Theory and practice* (5th ed.). Pearson.

SGOU



UNIT 4 Project Reviews and Agile Practices

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the purpose of Project Launch Reviews and Project Assessments
- describe the major Agile meetings used during software development
- apply Agile review practices to monitor project progress and obtain feedback
- analyse the role of Agile meetings and reviews in continuous software development
- analyse the relationship between Agile practices and suitable business models

Prerequisite

Consider a software company that is about to begin developing an online learning platform. Before development starts, the project team needs to confirm whether the project objectives are clear, the required resources are available, major risks have been considered, and the project plan is realistic. A Project Launch Review can help the organisation examine these aspects before development begins. As the project progresses, Project Assessments can be conducted to determine whether the project is achieving its planned objectives. Therefore, learners should have a basic understanding of software engineering, software project management, software quality, software reviews, software development life cycle, and Agile software development before studying this unit.

Studying Project Reviews and Agile Practices helps learners understand how software projects can be monitored and improved throughout their life cycle. Knowledge of project reviews helps in identifying risks, delays, resource problems, and quality issues at an early stage. Agile meetings and reviews help teams communicate regularly, obtain customer feedback, adapt to changes, and improve their development process. Understanding Agile business models also helps learners connect software development activities with customer value and business objectives.

SGOU

Keywords

Project Review, Project Launch Review, Project Assessment, Project Readiness, Project Progress, Agile, Agile Meeting, Daily Stand-Up, Sprint Planning, Sprint Review, Sprint Retrospective, Agile Review, Customer Feedback, Continuous Improvement, Business Model, Incremental Delivery, Business Value

Discussion

2.4.1 Introduction

Software projects require continuous monitoring and review to ensure that they remain aligned with their objectives, requirements, schedule, budget, and quality expectations. A project may begin with clearly defined goals, but changes in requirements, resources, technology, risks, and customer expectations can affect its progress. Therefore, project reviews and assessments are conducted at different stages of the project to identify problems, evaluate progress, and support appropriate decision-making.

Traditional project reviews are often conducted at important stages such as project initiation, planning, development, and completion. In Agile software development, review and communication activities are more frequent and are integrated into the development process. Agile teams conduct regular meetings, reviews, and reflection activities to understand progress, obtain feedback, and improve their way of working. Agile practices also encourage organisations to consider suitable business models that can support continuous delivery of customer value.

This unit discusses Project Launch Reviews, Project Assessments, Agile Meetings, Agile Reviews, and Agile Business Models. These concepts help learners understand how software projects can be reviewed, assessed, and continuously improved while maintaining focus on customer needs and business objectives.

2.4.2 Project Launch Review

A Project Launch Review is a review conducted at or near the beginning of a software project to determine whether the project is properly prepared for execution.

It provides an opportunity to verify that the project has clearly defined objectives, appropriate resources, realistic plans, and suitable management arrangements.

The Project Launch Review can help answer questions such as:

- Are the project objectives clearly defined?
- Are the requirements sufficiently understood?
- Are the required resources available?
- Is the project schedule realistic?

SGOU

- Have major risks been identified?
- Are responsibilities clearly assigned?
- Are quality expectations defined?
- Is the project ready to begin development?

2.4.2.1 Objectives of Project Launch Review

The major objectives are:

- To confirm project objectives.
- To verify project scope.
- To examine project feasibility.
- To verify resource availability.
- To identify major risks.
- To review the initial project schedule.
- To clarify roles and responsibilities.
- To establish quality expectations.
- To identify major dependencies.
- To determine project readiness.

2.4.2.2 Project Launch Review Process

A Project Launch Review can be performed through the following steps.

Step 1: Review Project Objectives

The team examines the purpose and expected outcomes of the project.

Step 2: Review Project Scope

The features and activities included and excluded from the project are examined.

Step 3: Review Requirements

The available requirements are examined for clarity, completeness, and consistency.

Step 4: Review Resources

Human resources, hardware, software, tools, infrastructure, and budget are considered.

Step 5: Identify Risks

Technical, financial, schedule, resource, security, and other major risks are identified.

SGOU

Step 6: Review the Schedule

The planned activities and deadlines are examined to determine whether they are realistic.

Step 7: Review Responsibilities

The responsibilities of project team members are clarified.

Step 8: Determine Readiness

The review team determines whether the project is ready to proceed.

2.4.3 Project Assessment

A **Project Assessment** is a systematic evaluation of the current status, performance, quality, risks, and progress of a software project.

Unlike a Project Launch Review, which mainly focuses on readiness at the beginning of a project, a Project Assessment can be conducted during the project to determine whether the project is progressing as expected.

Project Assessment can examine:

- Project progress
- Schedule performance
- Cost performance
- Quality
- Requirements
- Risks
- Resources
- Team performance
- Deliverables
- Customer satisfaction

2.4.3.1 Objectives of Project Assessment

The major objectives of Project Assessment are:

- To measure project progress.
- To identify delays.
- To evaluate project performance.

SGOU

- To identify risks and problems.
- To evaluate software quality.
- To examine resource utilisation.
- To determine whether requirements are being satisfied.
- To support management decisions.
- To identify corrective actions.
- To improve future project performance.

2.4.3.2 Project Assessment Process

A typical Project Assessment can involve the following steps:

Step 1: Collect Project Information

Relevant information about project progress, cost, schedule, quality, and deliverables is collected.

Step 2: Compare Planned and Actual Progress

The current project status is compared with the original plan.

Step 3: Examine Quality

The quality of deliverables and development activities is evaluated.

Step 4: Review Risks

Existing and newly identified risks are examined.

Step 5: Evaluate Resources

The utilisation and availability of team members, tools, and infrastructure are considered.

Step 6: Identify Problems

Major deviations, delays, defects, and other issues are identified.

Step 7: Recommend Corrective Actions

Suitable actions are proposed to address identified problems.

Step 8: Follow-up

The implementation and effectiveness of corrective actions are monitored.

SGOU

2.4.4 Project Launch Review and Project Assessment

Both activities are important but serve different purposes.

Table 2.4.1 Comparison of Project Launch Review and Project Assessment

Feature	Project Launch Review	Project Assessment
Main purpose	Determine project readiness	Evaluate project progress
Timing	Beginning of project	During or at important stages
Main focus	Planning and readiness	Performance and progress
Requirements	Initial understanding	Current status
Resources	Availability	Utilisation and adequacy
Risks	Initial risks	Existing and emerging risks
Schedule	Initial schedule	Actual versus planned progress
Outcome	Decision to proceed or modify plans	Corrective actions or improvements

2.4.5 Agile Software Development

Agile software development is an approach to software development that emphasises iterative development, customer collaboration, frequent delivery, continuous feedback, and responsiveness to changing requirements.

Instead of developing the entire system at once, Agile teams generally develop software in small increments.

For example, while developing an online shopping application, an Agile team may first develop:

1. User registration
2. Product search
3. Shopping cart
4. Payment
5. Order tracking

Each feature or group of features can be developed, reviewed, and improved through repeated development cycles.

SGOU

2.4.6 Agile Meetings

Agile teams use regular meetings to communicate, coordinate work, identify problems, and improve the development process.

Common Agile meetings include:

- Daily Stand-Up- A Daily Stand-Up is a short, regular team meeting used to communicate progress, identify problems, and coordinate daily work.
- Sprint Planning-Sprint Planning is a meeting in which the Agile team determines the work to be completed during the upcoming Sprint.
- Sprint Review-A Sprint Review is a meeting held at the end of a Sprint to examine the completed work and obtain feedback from stakeholders.
- Sprint Retrospective-A Sprint Retrospective is a meeting in which the Agile team reflects on its working process and identifies improvements for future Sprints.

These meetings are commonly associated with Scrum-based Agile development.

2.4.7 Agile Reviews

Agile Reviews are regular activities used to examine completed work, obtain feedback, and determine whether the product is moving in the desired direction.

The Sprint Review is an important example of an Agile review.

Agile reviews focus on:

- Customer value
- Completed features
- Product quality
- Feedback
- Changing requirements
- Future priorities

Agile reviews differ from traditional reviews because they are frequently integrated into short development cycles.

SGOU

2.4.8 Agile Meetings and Reviews

Table 2.4.2 Comparison of Major Agile Meetings

Meeting	Main Purpose	Typical Timing
Daily Stand-Up	Coordinate daily work	Daily
Sprint Planning	Plan Sprint work	Beginning of Sprint
Sprint Review	Demonstrate completed work and obtain feedback	End of Sprint
Sprint Retrospective	Improve team process	End of Sprint

2.4.9 Agile Business Models

A Business Model describes how an organisation creates, delivers, and captures value.

In software development, an Agile business model should support rapid delivery, customer feedback, continuous improvement, and adaptation to changing market requirements.

Traditional software projects may attempt to define most requirements at the beginning and deliver the complete product after a long development period. Agile approaches encourage organisations to deliver useful product increments frequently and learn from customer feedback.

2.4.9.1 Characteristics of Agile Business Models

Important characteristics include:

- **Customer Focus:** The business model gives importance to customer needs and feedback.
- **Continuous Value Delivery:** Useful features are delivered regularly rather than waiting until the entire product is completed.
- **Adaptability:** The organisation can respond to changing customer and market requirements.
- **Incremental Development:** Products and services can be developed and improved through smaller increments.
- **Continuous Feedback:** Customer and stakeholder feedback is regularly collected and used.

SGOU

- **Rapid Decision-Making:** Teams are encouraged to make decisions quickly based on available information.
- **Continuous Improvement:** The organisation continuously improves both its products and processes.

2.4.10 Traditional and Agile Business Approaches

Table 2.4.3 Comparison of Traditional and Agile Approaches

Feature	Traditional Approach	Agile Approach
Delivery	Often concentrated toward the end	Frequent incremental delivery
Requirements	Usually defined early	Can evolve during development
Customer feedback	Often less frequent	Frequent
Change	More difficult to accommodate	Expected and managed
Planning	Detailed long-term planning	Iterative planning
Product improvement	Mainly after major releases	Continuous
Decision-making	Often centralised	More collaborative
Business value	Often realised later	Delivered incrementally

Recap

- A Project Review is a systematic examination of project progress and performance.
- A Project Launch Review determines whether a project is ready to begin.
- Project Launch Reviews examine objectives, scope, requirements, resources, risks, and schedules.
- A Project Assessment evaluates the current status and performance of a project.
- Project Assessments help identify delays, risks, quality problems, and resource issues.
- Corrective actions can be taken based on Project Assessment findings.
- Agile development emphasises iterative development, customer collaboration, and continuous feedback.

SGOU

- Daily Stand-Up is used for daily communication and coordination.
- Sprint Planning is used to plan the work for an upcoming Sprint.
- Sprint Review is used to demonstrate completed work and obtain stakeholder feedback.
- Sprint Retrospective focuses on improving the team's development process.
- Agile reviews provide regular opportunities to examine completed work and adjust future priorities.
- An Agile Business Model focuses on customer value, adaptability, and continuous delivery.
- Agile practices help organisations respond to changing customer and market requirements.
- Project Reviews and Agile Practices support better project control, software quality, customer satisfaction, continuous improvement, and business value delivery.

Objective Type Questions

1. Which review is conducted at the beginning of a software project?
2. Name the process used to evaluate project progress and performance.
3. Identify the Agile meeting conducted on a daily basis.
4. Which meeting is used to plan the work of an upcoming Sprint?
5. Name the meeting in which completed work is demonstrated to stakeholders.
6. Which Agile meeting focuses on improving the team's working process?
7. Define the development approach that emphasises iterative development and continuous feedback.
8. Name the practice of delivering useful product features in small increments.
9. Identify the type of feedback regularly collected in Agile development.
10. Which assessment activity is used to identify and evaluate project risks?
11. Name the Agile meeting that mainly focuses on the completed product increment.

SGOU

12. Which Agile activity primarily focuses on process improvement?

13. Identify the model that explains how an organisation creates, delivers, and captures value.

Answers to Objective Type Questions

1. Project Launch Review.
2. Project Assessment.
3. Daily Stand-Up.
4. Sprint Planning.
5. Sprint Review.
6. Sprint Retrospective.
7. Agile Development.
8. Incremental Delivery.
9. Customer Feedback.
10. Risk Assessment.
11. Sprint Review.
12. Retrospective.
13. Business Model.

SGOU

Assignments

1. Explain the purpose of a Project Launch Review.
2. Describe the process of Project Assessment.
3. Explain the major Agile meetings used in software development.
4. Describe the characteristics of an Agile Business Model.
5. Explain the importance of Agile Reviews in software development.
6. Analyse the differences between Project Launch Review and Project Assessment.
7. Analyse how Agile meetings and reviews support continuous improvement in software development.

Suggested Reading

1. Jalote, P. (2025). *A concise introduction to software engineering: With open source and GenAI* (2nd ed.). Springer.
2. Sethi, R. (2023). *Software engineering: Basic principles and best practices*. Cambridge University Press.
3. Pressman, R. S., & Maxim, B. R. (2023). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.
4. O'Regan, G. (2022). *Concise guide to software engineering: From fundamentals to application methods*. Springer.
5. Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The definitive guide to Scrum: The rules of the game*. Scrum.org.

SGOU

Reference

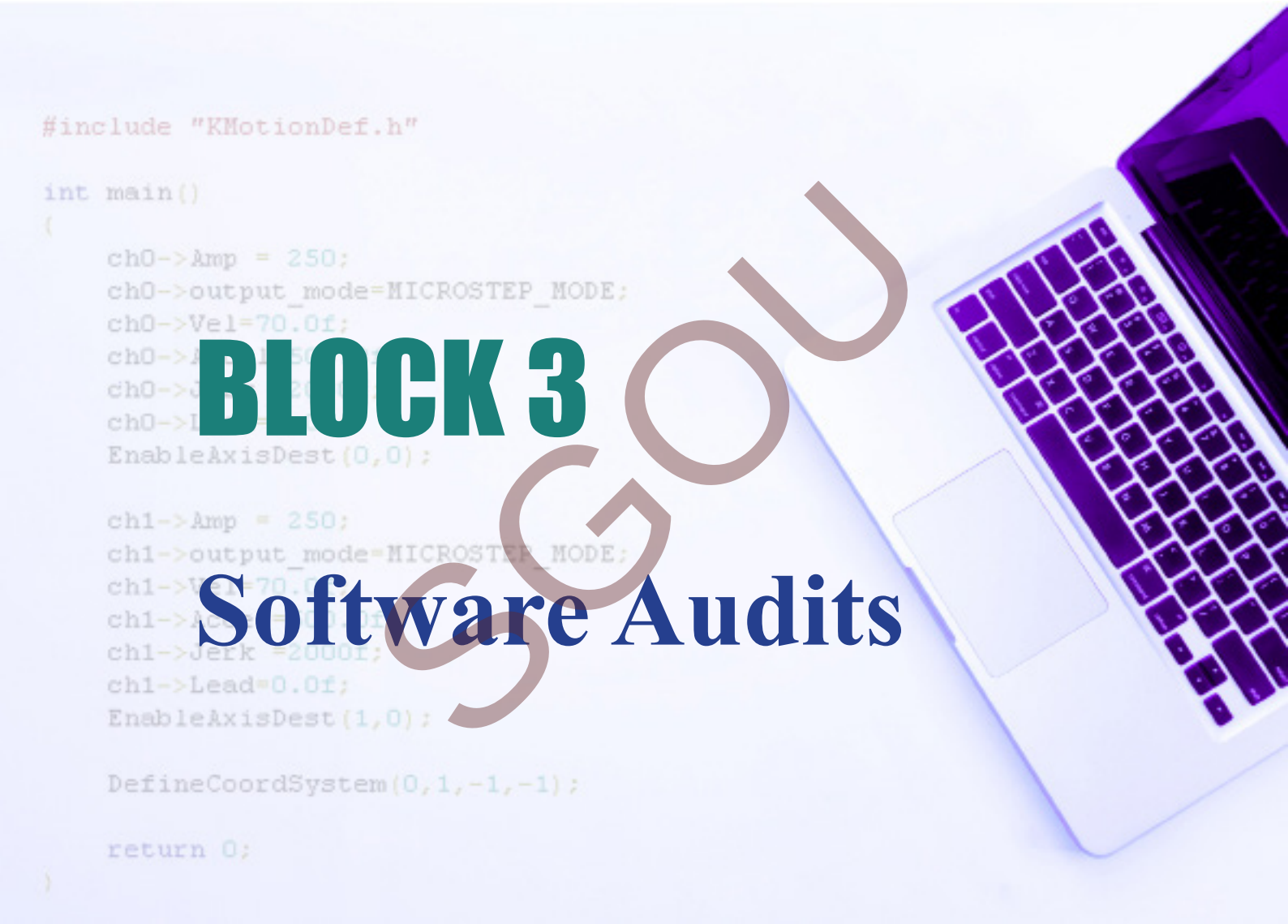
1. Dong, H., Dacre, N., Baxter, D., & Ceylan, S. (2024). What is Agile project management? Developing a new definition following a systematic literature review. *Project Management Journal, 55*(6), 668–688.

2. Radaideh, M. (2024). *Software project management: With PMI, IEEE-CS, and Agile-SCRUM*. De Gruyter.

3. Suvvari, S. K., & Jarubula, R. (2024). *Project management and software engineering*. Academic Guru Publishing House.

4. Kung, D. C. (2023). *Software engineering: An Agile unified methodology* (2nd ed.). McGraw Hill.

SGOU



BLOCK 3

Software Audits

SGOU



Types of Audits

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept and importance of software audits
- describe the different types of software audits used in software development
- differentiate between internal, external, product, process, system, compliance, quality, and security audits
- describe the objectives of different software audits
- explain the significance of software audits in ensuring software quality and compliance

Prerequisite

In your earlier studies, you have learned about the Software Development Life Cycle (SDLC), software testing, software quality, and the importance of developing reliable software systems. You are also familiar with activities such as requirement analysis, design, coding, testing, and maintenance. These activities help ensure that software is developed correctly and meets user requirements.

However, developing software alone is not enough. Organisations must also verify whether the software product, development processes, and quality practices follow established standards and produce the expected results. This is achieved through software audits. Software audits help identify weaknesses, ensure compliance with standards, improve software quality, and enhance customer confidence.

In this unit, you will learn the concept of software audit, its objectives and benefits, and the different types of software audits, including internal, external, product, process, system, compliance, quality, and security audits. Understanding these audits will help you appreciate how organisations maintain software quality, reliability, and security throughout the software development process.

SGOU

Keywords

Software Audit, Internal Audit, External Audit, Quality Audit, Security Audit

Discussion

3.1.1 What is Software Audit?

Software development involves a series of activities such as requirement analysis, design, coding, testing, and maintenance. During these activities, organisations need to ensure that the software product and the processes used to develop it conform to specified standards, procedures, and customer requirements. This verification is performed through a software audit.

A software audit is a systematic and independent examination of software products, processes, documents, and activities to determine whether they comply with predefined standards, guidelines, and requirements. It helps organisations evaluate the effectiveness of their software development practices and identify areas that require improvement.

In simple terms, a software audit is similar to a health check-up for a software project. Just as a doctor examines a person's health to detect problems and suggest corrective measures, a software audit examines the software and its development process to ensure that quality requirements are being met.

Software audits are conducted to answer questions such as:

- Are software development activities being carried out according to established procedures?
- Does the software product satisfy customer requirements?
- Are quality standards being followed?
- Are security measures adequately implemented?
- Are there any defects or non-conformities that need corrective action?

The primary purpose of a software audit is not to find faults or blame individuals, but to ensure that software products and processes are effective, reliable, and continuously improved.

Example

Consider a company developing an Online Shopping System. Before releasing the software to customers, the organisation conducts a software audit. During the audit, the auditors examine:

- Whether all requirements have been implemented.
- Whether coding standards have been followed.
- Whether sufficient testing has been performed.

SGOU

- Whether user manuals and technical documents are complete.
- Whether security measures have been incorporated.

Based on the findings, the organisation takes necessary corrective actions to improve the quality of the software before deployment.

Thus, software audits play an important role in producing high-quality software, reducing risks, ensuring compliance with standards, and increasing customer satisfaction.

3.1.2 Need and Importance of Software Audits

In the previous section, we learned that a software audit is a systematic examination of software products, processes, documents, and activities to determine whether they conform to specified requirements and standards. However, an important question arises: Why are software audits necessary?

As software systems become increasingly complex and are used in critical areas such as banking, healthcare, education, and e-commerce, ensuring their reliability, security, and quality has become essential. Software audits help organisations evaluate their development practices, identify deficiencies, and implement corrective actions. Thus, software audits have become an important tool for maintaining software quality and supporting continuous process improvement.

The importance of software audits can be understood from the following perspectives.

1. **Ensuring Software Quality:** One of the primary objectives of a software audit is to ensure that software products satisfy specified quality requirements. Audits verify whether the software performs its intended functions correctly and whether development activities follow established standards and procedures.

   For example, during the development of an online examination system, an audit may reveal that certain test cases have not been executed. Corrective actions can then be taken before the software is released, thereby improving its quality.

2. **Identifying Errors and Deficiencies:** Software audits help identify defects, inconsistencies, and deviations from prescribed procedures. Detecting such problems at an early stage prevents minor issues from becoming major failures and reduces the cost of correction.

   For instance, an audit of a payroll management system may reveal incomplete documentation or insufficient testing. These shortcomings can be rectified before the software is deployed.

3. **Ensuring Compliance with Standards and Regulations:** Organisations are often required to comply with international standards, government regulations, and contractual obligations. Software audits verify whether these requirements are being followed.

SGOU

For example, banking applications must adhere to security policies and data protection regulations. Audits help ensure that these requirements are properly implemented.

4. **Improving Development Processes:** Software audits do not focus only on the final product; they also evaluate the processes used to develop the software. They help organisations identify weaknesses in existing procedures and recommend improvements.

   For example, if auditors observe that requirement changes are not documented properly, the organisation can establish better change management practices to enhance process efficiency.

5. **Reducing Project Risks:** Software projects are exposed to various risks, including schedule delays, cost overruns, software failures, and security vulnerabilities. Software audits help identify these risks at an early stage and enable organisations to take preventive measures.

   For example, a security audit may detect weak password policies in an online banking application. By strengthening authentication mechanisms, the organisation can reduce the risk of unauthorised access.

6. **Increasing Customer Confidence:** Customers expect software products to be reliable and free from major defects. Regular audits demonstrate an organisation's commitment to quality and increase customer trust and confidence.

   For example, organisations possessing quality certifications such as ISO 9001 provide assurance to customers that their software development activities follow internationally accepted standards.

7. **Supporting Continuous Improvement:** Continuous improvement is essential for organisations to remain competitive in a rapidly changing technological environment. Audit findings provide valuable feedback that enables organisations to enhance their processes and products.

As a result, organisations can achieve higher productivity, better software quality, and increased customer satisfaction.

3.1.3 Types of Software Audits

Software audits play a significant role in ensuring software quality, reducing risks, and promoting continuous improvement. However, a software project involves several activities, including requirements analysis, design, coding, testing, documentation, and maintenance. Since each of these activities has different objectives and requirements, a single audit may not be sufficient to evaluate all aspects of software development.

Therefore, organisations conduct different types of audits depending on the purpose and scope of evaluation. Some audits focus on the software product itself, while others examine the processes followed during development. Certain audits are intended to ensure compliance with standards and regulations, whereas others concentrate on

SGOU

quality and security aspects. Each type of audit contributes to improving the overall effectiveness and reliability of software systems. The major types of software audits are illustrated in Figure 3.1.1.



Fig. 3.1.1 Types of Software Audits

3.1.3.1 Internal Audit

An internal audit is an audit conducted by employees within an organisation to examine whether software development activities are being carried out according to the organisation's policies, procedures, and quality standards. It is usually performed by the Software Quality Assurance (SQA) team or trained employees who are independent of the project being audited.

The main purpose of an internal audit is to identify weaknesses in the software development process and recommend improvements before the software is delivered to the customer. Since the audit is conducted internally, it helps the organisation detect problems at an early stage and take corrective actions before they become major issues. Unlike external audits, internal audits are not conducted for certification but for improving the overall quality and efficiency of software development activities.

Internal audits are generally carried out at different stages of the Software Development Life Cycle (SDLC), such as during requirement analysis, design, coding, testing, and maintenance. During the audit, auditors review project documents, examine software development activities, verify records, and interact with project team members to ensure that organisational procedures and quality standards are being followed.

For example, suppose a software company is developing a Hospital Management System. During an internal audit, the audit team reviews the Software Requirements Specification (SRS), design documents, source code, test reports, and user manuals. While reviewing the project, the auditors discover that some testing activities have not been completed and that a few modules do not follow the organisation's coding standards. These observations are recorded in the audit report, and the development team is instructed to complete the pending testing activities and modify the code according to the prescribed standards. After implementing these corrections, the software becomes more reliable and ready for customer delivery.

SGOU

Internal audits offer several advantages to software organisations. They help identify defects at an early stage, improve software quality, ensure compliance with organisational standards, reduce development risks, and encourage continuous process improvement. Since problems are identified before the software is released, the cost of correcting defects is significantly lower than fixing them after deployment. Internal audits also prepare the organisation for external audits conducted by customers or certification agencies.

Although internal audits provide many benefits, they may have certain limitations. Since the auditors are employees of the same organisation, complete independence may not always be possible. In some cases, personal bias or familiarity with existing practices may influence the audit findings. However, these limitations can be minimised by selecting trained auditors who are not directly involved in the project being audited.

Features of Internal Audit

The important features of an internal audit in software development are as follows:

1. **Continuous Process:** Internal audits are conducted regularly throughout the Software Development Life Cycle (SDLC) rather than only at the end of a project. Periodic audits help identify problems early, making them easier and less expensive to correct.

2. **Conducted by Internal Personnel:** Internal audits are usually performed by the organisation's Software Quality Assurance (SQA) team or trained employees who are independent of the project being audited. In some cases, organisations may also appoint independent internal auditors to ensure objectivity.

3. **Evaluates Software Processes and Controls:** An internal audit examines whether software development activities, quality procedures, coding standards, testing practices, and documentation comply with the organisation's established policies and standards.

4. **Improvement-Oriented:** The primary purpose of an internal audit is to improve software development processes rather than to find faults or assign blame. The audit findings help the organisation implement corrective and preventive actions for continuous improvement.

5. **Covers All Phases of Software Development:** Internal audits may be conducted during different phases of the Software Development Life Cycle, including requirement analysis, system design, coding, testing, deployment, and maintenance, ensuring that quality is maintained throughout the project.

6. **Helps Identify Defects and Risks:** By continuously reviewing software activities, internal audits help identify defects, process deviations, documentation issues, and potential project risks before the software is released to users.

7. **Flexible Scope:** The scope of an internal audit depends on the organisation's objectives and the nature of the software project. It may cover the entire project or focus on specific activities such as coding, testing, configuration management, or documentation.

SGOU

Process of Internal Audit

An internal audit follows a systematic procedure to evaluate whether software development activities comply with the organisation's policies, procedures, and quality standards. Although the exact procedure may vary from one organisation to another, the internal audit process generally consists of the following steps.



Fig. 3.1.2 Process flow of Internal Audit

1. Audit Planning

The first step in an internal audit is planning. During this stage, the audit team determines the objectives, scope, and schedule of the audit. The areas to be examined, such as requirements, design, coding, testing, or documentation, are identified. Necessary documents and resources required for the audit are also collected.

Proper planning ensures that the audit is organised and covers all important aspects of the software project.

2. Review of Documents

After planning, the auditors review the documents related to the software project. These may include the Software Requirements Specification (SRS), design documents, project plans, coding standards, test reports, configuration management records, and user manuals.

The purpose of this review is to verify whether the documents are complete, accurate, and prepared according to organisational standards.

3. Examination of Software Development Activities

In this stage, auditors examine the actual software development activities. They observe how the project team performs tasks such as coding, testing, version control, defect

SGOU

management, and configuration management. They may also interview developers, testers, and project managers to understand how the work is being carried out.

The objective is to determine whether the development process complies with the organisation's established procedures and quality standards.

4. Identification of Audit Findings

Based on the review and examination, auditors identify any non-conformities, defects, missing documents, or deviations from the prescribed standards. They also recognise good practices that contribute to software quality.

The findings are documented with appropriate evidence and classified according to their severity so that management can prioritise corrective actions.

5. Preparation of Audit Report

After completing the audit, the auditors prepare a detailed audit report. The report summarises the observations, strengths, weaknesses, non-conformities, and recommendations for improvement.

The audit report is submitted to the project manager or management team for review and further action.

6. Corrective Actions

The project team reviews the audit findings and implements corrective actions to resolve the identified issues. This may involve updating documents, modifying source code, improving testing procedures, or revising development practices.

Corrective actions help eliminate existing problems and reduce the likelihood of similar issues occurring in future projects.

7. Follow-up Audit

The final step is the follow-up audit. During this stage, auditors verify whether the recommended corrective actions have been implemented effectively. If necessary, additional recommendations are provided to ensure continuous improvement.

The follow-up process confirms that the identified problems have been resolved and that the software development process complies with the required standards.

SGOU

Advantages and Disadvantages of Internal Audit

Table 3.1.1 Advantages & Disadvantages of Internal Audit

Advantages	Disadvantages
Helps identify problems at an early stage	May not be completely independent because it is conducted by employees of the organisation
Improves the quality of the software	Personal bias may affect the audit results
Ensures that software development follows organisational standards and procedures	Requires additional time and resources
Helps improve software development processes	Some issues may be overlooked due to familiarity with the project

3.1.3.2 External Audit

While an internal audit is conducted by employees within the organisation, there are situations where an independent evaluation is required. Customers, certification bodies, or regulatory authorities may need assurance that the organisation's software development activities comply with established standards and quality requirements. Such an independent evaluation is called an external audit.

An external audit is conducted by auditors who are not part of the organisation. These audits are usually carried out by certification agencies, customers, or government authorities. Since the auditors are independent, they provide an objective and unbiased assessment of the organisation's software products, development processes, and quality management practices.

The main purpose of an external audit is to verify whether the organisation follows international standards, legal requirements, and contractual obligations. External audits are commonly conducted when an organisation applies for certifications such as ISO 9001 or when customers assess the organisation's ability to deliver quality software.

During the audit, the auditors review project documents, quality records, testing reports, and software development procedures. They may also interview project team members to understand how the work is carried out. Based on their observations, the auditors identify any non-conformities and recommend corrective actions to improve compliance and software quality.

Features of External Audit

The important features of an external audit are as follows:

1. **Conducted by Independent Auditors:** External audits are performed by auditors who are not employees of the organisation. This ensures that the audit is fair, impartial, and objective.

SGOU

2. **Focuses on Compliance:** The audit verifies whether the organisation follows applicable standards, legal requirements, contractual obligations, and quality management practices.

3. **Independent Assessment:** Since the auditors are independent, the audit results are considered more reliable by customers, certification bodies, and regulatory authorities.

4. **Supports Certification:** External audits are often conducted as part of the certification process for standards such as ISO 9001 or other quality management systems.

5. **Periodic Evaluation:** External audits are usually conducted at scheduled intervals or whenever required by customers, certification agencies, or regulatory bodies.

6. **Builds Customer Confidence:** A successful external audit demonstrates that the organisation follows recognised quality practices, thereby increasing customer trust and organisational credibility.

Process of External Audit

An external audit generally follows the steps as shown in the figure 3.1.4.



Fig. 3.1.4 Process flow of External Audit

SGOU

1. Audit Planning

The external auditors define the objectives, scope, audit schedule, and evaluation criteria. The organisation is informed about the audit plan and the documents required.

2. Review of Documents

The auditors examine quality manuals, project plans, requirement documents, design documents, coding standards, testing reports, and other relevant records to determine whether they comply with the specified standards.

3. On-site Audit

The auditors visit the organisation, observe software development activities, interview employees, and verify whether established procedures are being followed.

4. Identification of Findings

The auditors identify any non-conformities, observations, or opportunities for improvement and record them as audit findings.

5. Audit Report

A detailed audit report is prepared summarising the findings, non-conformities, strengths, and recommendations.

6. Corrective Actions

The organisation implements corrective actions to resolve the identified issues and improve compliance.

7. Certification or Follow-up

If the organisation satisfies all the requirements, the certification body may grant certification. If major non-conformities exist, a follow-up audit may be conducted after corrective actions have been completed.

3.1.3.3 Product Audit

Internal and external audits mainly evaluate the software development process, quality management practices, and compliance with organisational or international standards. However, after the software has been developed, it is equally important to verify whether the final software product satisfies the specified requirements and quality standards. This evaluation is carried out through a product audit.

A product audit is a systematic examination of a software product to determine whether it conforms to the specified requirements, design specifications, quality standards, and customer expectations. Unlike process audits, which focus on *how* the software is developed, a product audit focuses on *what* has been developed. It evaluates the completed software product to ensure that it functions correctly, is reliable, and is ready for delivery to the customer.

SGOU

A product audit is usually conducted after the completion of software development or before the software is released. The audit helps verify that the software meets functional and non-functional requirements, complies with coding and testing standards, and is free from major defects. It also confirms that all required project documents, user manuals, and test reports are complete and up to date.

By conducting a product audit, organisations can identify any remaining issues in the software product before deployment. This helps improve software quality, increase customer satisfaction, and reduce the chances of software failures after release.

Features of Product Audit

The important features of a product audit are as follows:

- **Focuses on the Final Software Product:** A product audit examines the completed software product rather than the development process.
- **Verifies Customer Requirements:** It checks whether the software meets all the functional and non-functional requirements specified by the customer.
- **Evaluates Software Quality:** The audit verifies important quality attributes such as functionality, reliability, usability, performance, and maintainability.
- **Reviews Supporting Documents:** In addition to the software, the audit also examines user manuals, technical documentation, installation guides, and test reports to ensure completeness and accuracy.
- **Conducted Before Software Release:** A product audit is generally performed before the software is delivered to the customer or deployed in the production environment.
- **Helps Improve Customer Satisfaction:** By identifying defects before delivery, the audit ensures that customers receive a reliable and high-quality software product.

Example:

A software company develops an Online Shopping System for an e-commerce business. Before delivering the software, a product audit is conducted. The auditors verify whether customers can register, search for products, add items to the shopping cart, make secure online payments, and track their orders. They also review the user manual, installation guide, and testing reports.

During the audit, it was found that the payment module occasionally fails under heavy user traffic, and the user manual lacks instructions for password recovery. These issues are corrected before the software is delivered. As a result, the customer receives a reliable and well-documented software product.

SGOU

Advantages and Disadvantages of Product Audit

Table 3.1.2 Advantages and Disadvantages of Product Audit

Advantages	Disadvantages
Ensures that the software meets customer requirements	Conducted mainly after product development, so major changes may be costly
Helps identify defects before software release	Can increase the project completion time
Improves the quality and reliability of the software	Requires skilled auditors to evaluate the product effectively
Ensures that documentation is complete and accurate	May not identify problems in the development process
Increases customer satisfaction and confidence.	Additional testing and review may increase project costs
Reduces the chances of software failure after deployment.	Large software products may require more time for auditing

3.1.3.4 Process Audit

A product audit evaluates the quality of the completed software product. However, the quality of a software product largely depends on the processes followed during its development. Even if the final product appears to function correctly, weaknesses in the development process can lead to quality issues, delays, or increased maintenance costs in future projects. To ensure that software is developed using effective and standardised procedures, organisations conduct a process audit.

A process audit is a systematic examination of the software development processes to determine whether they are being carried out according to the organisation's policies, procedures, quality standards, and best practices. Instead of evaluating the completed software product, a process audit focuses on how the software is developed. It examines whether each phase of the Software Development Life Cycle (SDLC), such as requirement analysis, design, coding, testing, deployment, and maintenance, follows the prescribed procedures.

The main purpose of a process audit is to verify that software development activities are planned, documented, and executed consistently. It helps organisations identify weaknesses in their development processes and implement corrective actions to improve efficiency, quality, and productivity. Process audits also encourage continuous improvement by ensuring that lessons learned from previous projects are applied to future projects.

Features of Process Audit

The important features of a process audit are as follows:

- **Focuses on Development Processes:** A process audit examines the methods, procedures, and activities followed during software development rather than the final software product.

SGOU

- **Verifies Compliance with Standards:** It checks whether the organisation follows established software development standards, coding guidelines, testing procedures, and quality policies.
- **Covers the Entire SDLC:** The audit may evaluate activities performed during requirement analysis, system design, coding, testing, deployment, and maintenance.
- **Promotes Process Improvement:** The audit identifies weaknesses in existing processes and recommends improvements to increase efficiency and software quality.
- **Encourages Consistency:** Process audits help ensure that similar projects follow uniform development procedures across the organisation.
- **Reduces Project Risks:** By identifying process-related issues early, the audit helps reduce delays, defects, and rework in software projects.

Example

A software company is developing a Student Information Management System. During a process audit, the auditors review project plans, coding standards, testing procedures, and configuration management practices. They observe that developers are not conducting code reviews before integrating new modules into the system. They also find that requirement changes are not being documented properly.

The auditors recommend introducing mandatory code reviews and maintaining a formal change management process. After implementing these recommendations, the organisation improves the consistency and quality of its software development process, leading to fewer defects and better project management.

3.1.3.5 System Audit

A process audit evaluates the activities and procedures followed during software development. However, the successful operation of a software application also depends on the overall system in which it is used. This includes hardware, software, databases, networks, security mechanisms, and operating procedures. To ensure that all these components work together efficiently and support the organisation's objectives, a system audit is conducted.

A system audit is a systematic examination of an entire computer-based system to determine whether it operates effectively, securely, and in accordance with organisational requirements and established standards. Unlike a product audit, which focuses on the software product, or a process audit, which evaluates the development process, a system audit evaluates the complete information system and its supporting infrastructure.

The primary purpose of a system audit is to ensure that all components of the system function correctly, protect organisational data, and provide reliable services to users. During the audit, auditors examine system configuration, hardware resources, software applications, databases, network infrastructure, backup and recovery procedures, and security controls. They also verify whether the system is capable of meeting business requirements and supporting day-to-day operations without major failures.

SGOU

System audits are especially important for organisations that rely heavily on information systems, such as banks, hospitals, universities, government departments, and e-commerce companies, where system failures can lead to significant financial and operational losses.

Features of System Audit

The important features of a system audit are as follows:

- **Evaluates the Entire System:** A system audit examines the complete information system, including hardware, software, databases, networks, and supporting resources.
- **Verifies System Performance:** The audit evaluates whether the system performs efficiently, reliably, and according to organisational requirements.
- **Examines Security Controls:** It checks whether appropriate security measures are in place to protect data and prevent unauthorised access.
- **Reviews Backup and Recovery Procedures:** The audit verifies whether backup mechanisms and disaster recovery plans are available and functioning properly.
- **Ensures Data Integrity:** A system audit confirms that data stored and processed by the system is accurate, complete, and protected from unauthorised modifications.
- **Supports Business Continuity:** The audit helps ensure that the system can continue operating effectively even during unexpected failures or emergencies.

Example

A university uses an Online Learning Management System (LMS) to conduct classes, assignments, and examinations. A system audit is carried out to evaluate the entire system. The auditors examine the application software, database server, network connectivity, backup procedures, and user authentication mechanisms.

During the audit, they discover that backup files are not created regularly and that some user accounts have unnecessary administrative privileges. The organisation immediately updates its backup policy and modifies user access permissions. As a result, the system becomes more secure, reliable, and capable of supporting uninterrupted online learning.

3.1.3.6 Compliance Audit

A system audit evaluates the overall performance, security, and reliability of an information system. However, organisations must also ensure that their software development activities comply with organisational policies, legal requirements, contractual obligations, and industry standards. This verification is carried out through a compliance audit.

A compliance audit is a systematic examination conducted to determine whether an organisation follows the rules, policies, standards, and regulations applicable to its software development activities. The audit verifies that the organisation complies with

SGOU

internal policies as well as external requirements such as government regulations, customer agreements, and international standards.

The primary purpose of a compliance audit is to identify any deviations from established requirements and ensure that corrective actions are taken. Compliance audits help organisations reduce legal and operational risks, improve accountability, and maintain customer confidence. These audits are commonly performed in organisations seeking certifications, working with government agencies, or handling sensitive information.

During a compliance audit, auditors review project documents, organisational policies, security procedures, quality records, and development practices. They compare these with the applicable standards and regulations to determine whether the organisation is meeting the required level of compliance.

Features of Compliance Audit

The important features of a compliance audit are as follows:

- **Focuses on Compliance:** The audit verifies whether the organisation follows applicable laws, regulations, standards, and internal policies.
- **Evaluates Organisational Practices:** It examines software development activities, documentation, and operational procedures to ensure they meet the required standards.
- **Identifies Non-Compliance:** The audit identifies areas where the organisation fails to meet specified requirements and recommends corrective actions.
- **Supports Legal and Regulatory Requirements:** Compliance audits help organisations meet legal obligations and avoid penalties resulting from non-compliance.
- **Encourages Standardisation:** The audit promotes the consistent implementation of organisational policies and industry best practices.
- **Builds Customer Confidence:** Demonstrating compliance with recognised standards increases customer trust and enhances the organisation's reputation.

3.1.3.7 Quality Audit

A compliance audit verifies whether an organisation follows the required standards, policies, and regulations. However, simply complying with these requirements does not always guarantee the delivery of high-quality software. Organisations must also evaluate whether their quality management practices are effective in producing reliable and defect-free software products. This evaluation is carried out through a quality audit.

A quality audit is a systematic examination of the software development process and quality management system to determine whether quality standards, procedures, and practices are being implemented effectively. It verifies that the organisation follows its quality policies and continuously works towards improving software quality.

SGOU

The primary purpose of a quality audit is to assess whether the quality management system is functioning as intended and whether software products meet the expected quality standards. During the audit, auditors examine quality plans, development procedures, testing activities, review reports, defect records, and other quality-related documents. They also verify whether quality assurance activities are carried out throughout the Software Development Life Cycle (SDLC).

By identifying weaknesses in quality management practices, a quality audit helps organisations improve software quality, reduce defects, and increase customer satisfaction.

Features of Quality Audit

The important features of a quality audit are as follows:

- **Focuses on Software Quality:** The audit evaluates whether software products and development activities meet the required quality standards.
- **Evaluates the Quality Management System:** It examines whether the organisation's quality policies, procedures, and practices are implemented effectively.
- **Verifies Quality Assurance Activities:** The audit checks whether activities such as reviews, testing, inspections, and defect management are performed properly.
- **Encourages Continuous Improvement:** Audit findings help organisations improve their quality management practices and software development processes.
- **Identifies Quality Issues:** The audit detects defects, process weaknesses, and areas requiring improvement before the software is released.

Improves Customer Satisfaction: By ensuring high software quality, the audit increases customer confidence and satisfaction.

3.1.3.8 Security Audit

A quality audit ensures that software products and development processes meet the required quality standards. However, high-quality software must also protect information from security threats such as unauthorized access, data theft, malware, and cyberattacks. To evaluate the effectiveness of security measures implemented in a software system, organisations conduct a security audit.

A security audit is a systematic examination of a software system to determine whether appropriate security controls are in place to protect the system and its data. It evaluates the confidentiality, integrity, and availability of information by identifying security vulnerabilities and verifying whether security policies and procedures are being followed.

The primary purpose of a security audit is to identify potential security risks and ensure that the software system is protected against internal and external threats. During the audit, auditors examine user authentication mechanisms, access control policies, password management, data encryption, network security, system logs, backup

SGOU

procedures, and other security-related measures. They also verify whether security controls comply with organisational policies and applicable standards.

A security audit helps organisations strengthen their security practices, reduce the risk of cyberattacks, and ensure the safe operation of software systems.

Features of Security Audit

The important features of a security audit are as follows:

1. **Focuses on System Security:** A security audit evaluates whether the software system is protected against security threats and vulnerabilities.

2. **Examines Security Controls:** The audit verifies the effectiveness of authentication, authorisation, encryption, firewall settings, and other security mechanisms.

3. **Identifies Security Vulnerabilities:** It helps detect weaknesses that could allow unauthorised access, data loss, or cyberattacks.

4. **Protects Sensitive Information:** The audit ensures that confidential information is stored, processed, and transmitted securely.

5. **Verifies Compliance with Security Policies:** The audit checks whether the organisation follows its security policies and applicable security standards.

6. **Supports Continuous Security Improvement:** The audit provides recommendations to strengthen security measures and reduce future risks.

Recap

- A software audit is a systematic examination of software products, processes, and activities.
- Software audits help improve software quality and ensure compliance with standards.
- Different types of software audits are conducted for different purposes.
- An internal audit is conducted by the organisation's own audit team.
- An external audit is conducted by independent auditors or certification bodies.
- A product audit evaluates the completed software product.
- A process audit examines the software development process.
- A system audit evaluates the overall information system and its components.
- A compliance audit verifies adherence to standards, policies, and regulations.

SGOU

- A quality audit assesses the effectiveness of quality management practices.
- A security audit identifies security risks and evaluates security controls.
- Software audits help organisations deliver reliable, secure, and high-quality software products.

Objective Type Questions

1. What is the systematic examination of software products, processes, and activities called?
2. Which type of audit is conducted by the organisation's own audit team?
3. Which type of audit is performed by independent auditors?
4. Which audit focuses on the completed software product?
5. Which audit evaluates the software development process?
6. Which audit examines the complete information system?
7. Which audit verifies adherence to standards, policies, and regulations?
8. Which audit assesses the effectiveness of quality management practices?
9. Which audit identifies security risks and evaluates security controls?
10. What document contains the findings and recommendations of an audit?

SGOU

Answers to Objective Type Questions

1. Software Audit.
2. Internal Audit.
3. External Audit.
4. Product Audit.
5. Process Audit.
6. System Audit.
7. Compliance Audit.
8. Quality Audit.
9. Security Audit.
10. Audit Report.

Assignments

1. Explain the concept of a software audit and discuss its importance in software development.
2. Describe the different types of software audits with suitable examples.
3. Differentiate between internal audit and external audit with the help of suitable examples.
4. Explain the process involved in conducting a software audit.
5. Discuss the role of quality audit and security audit in improving software quality and protecting software systems.

Suggested Reading

1. Hoyle, David. *ISO 9000 Quality Systems Handbook*. 7th ed., Routledge.
2. O'Regan, Gerard. *Introduction to Software Quality*. Springer, 2014.
3. International Organization for Standardization. *ISO/IEC 29110-1-1:2024: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 1-1: Overview*. ISO, 2024.
4. International Organization for Standardization. *ISO/IEC 29110-5-1-1:2025: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 5-1-1: Software Engineering Guidelines for the Generic Entry Profile*. ISO, 2025.

Reference

1. Galin, Daniel. *Software Quality: Concepts and Practice*. 2nd ed., Wiley-IEEE Computer Society Press, 2018.
2. O'Regan, Gerard. *Introduction to Software Process Improvement*. Springer, 2011.
3. International Organization for Standardization. *ISO 9001:2015: Quality Management Systems—Requirements*. ISO, 2015.

SGOU



Audit Process and the ISO 9001 Standard

Learning Outcomes

After completion of this unit, learner will be able to:

- define the audit process and its phases
- explain the different types of software audits
- describe the activities involved in audit planning and audit execution
- explain the concepts of ISO 9001 and Quality Management Systems
- describe the ISO 9001 certification process and its benefits

Prerequisite

Software organisations develop software by following a series of planned activities such as requirements analysis, design, coding, testing, and documentation. You have already learned about different types of software audits and their role in ensuring software quality. You also know that software products and development processes should comply with defined standards, organisational policies, and customer requirements.

Consider a software company developing an Online Banking System. After completing the development work, the organisation needs to verify whether the project followed the required quality standards and development procedures. The auditors must decide how the audit will be conducted, what documents should be reviewed, what evidence should be collected, and how the findings should be reported. A systematic audit process helps ensure that the audit is carried out in an organised and reliable manner.

This unit explains the audit process, its phases, audit planning, audit execution, and the ISO 9001 Quality Management System. You will also learn about the ISO 9001 certification process and understand how these practices help organisations improve software quality, ensure compliance with standards, and achieve continuous improvement.

Keywords

Audit Process, Audit Planning, Audit Execution, ISO 9001, ISO 9001 Certification

SGOU

Discussion

3.2.1 What is an Audit Process?

In the previous unit, you learned about the concept of software audits and the different types of audits used in software development. Understanding the different types of audits is important, but it is equally important to know how an audit is conducted. A software audit is not a random activity; instead, it follows a well-defined sequence of steps known as the audit process. Following a systematic process helps auditors collect accurate evidence, identify non-conformities, and prepare reliable audit reports.

An audit process is a structured and systematic sequence of activities carried out to evaluate whether a software product, software development process, or quality management system complies with specified requirements, standards, and organisational policies. It provides a clear framework for planning, conducting, documenting, and following up on an audit.

A well-defined audit process ensures that every audit is conducted in a consistent, objective, and organised manner. It helps auditors evaluate software products and development processes using a standard approach, making the audit results more reliable and meaningful. By following each stage of the audit process, organisations can identify weaknesses, improve software quality, ensure compliance with standards, and support continuous improvement.

The audit process generally consists of seven phases: Audit Initiation, Audit Planning, Audit Preparation, Audit Execution, Audit Reporting, Corrective Actions, and Audit Follow-up. Each phase has a specific purpose and contributes to the successful completion of the audit. These phases are discussed in the following sections.

3.2.2 Phases of the Audit Process

An audit process is carried out through a series of well-defined phases. Each phase has a specific purpose and helps ensure that the audit is conducted in a systematic, objective, and organised manner. By following these phases, auditors can effectively evaluate software products, software development processes, and quality management systems.

Although the purpose and scope of an audit may differ across organiations, the basic phases of the audit process are generally the same. These phases help auditors plan the audit, collect evidence, identify non-conformities, report findings, and ensure that corrective actions are taken.

SGOU

The audit process generally consists of the following seven phases:



Fig. 3.2.1 Phases of Audit Process

1. Audit Initiation

The audit initiation phase is the first stage of the audit process. In this phase, the organisation decides to conduct an audit based on business needs, customer requirements, regulatory requirements, or quality improvement initiatives. The purpose and scope of the audit are clearly defined to determine what will be audited and why the audit is necessary.

During this phase, the audit criteria are identified, an audit team is appointed, and the concerned departments are informed about the upcoming audit. A well-planned audit initiation phase provides a clear direction for the entire audit process.

Activities Performed During Audit Initiation

- Identify the need for the audit.
- Define the audit objectives and scope.
- Identify the audit criteria.
- Select the audit team.
- Inform the concerned departments.

SGOU

2. Audit Preparation

The audit preparation phase focuses on collecting and reviewing the information required before conducting the actual audit. During this phase, auditors study project documents, organisational policies, quality records, and previous audit reports to gain a clear understanding of the software project.

Proper preparation helps auditors perform the audit more effectively and ensures that all necessary information is available before the audit begins.

Activities Performed During Audit Preparation

- Review project documents.
- Study organisational policies.
- Examine previous audit reports.
- Prepare interview questions.
- Verify the availability of required resources.

3. Audit Execution

The audit execution phase is the stage in which the actual audit is conducted. During this phase, auditors collect evidence by reviewing documents, observing software development activities, interviewing employees, and examining project records. The collected evidence is compared with the audit criteria to determine whether the organisation complies with the specified requirements.

The findings recorded during this phase form the basis for the audit report.

Activities Performed During Audit Execution

- Conduct the opening meeting.
- Review documents and records.
- Observe work activities.
- Interview project personnel.
- Collect and record audit evidence.

4. Audit Reporting

After completing the audit, the auditors prepare an audit report that summarises the audit findings. The report includes observations, identified non-conformities, strengths, and recommendations for improvement. It provides management with a clear understanding of the audit results and supports decision-making.

A well-prepared audit report serves as an important reference for future audits and quality improvement activities.

SGOU

Activities Performed During Audit Reporting

- Compile the audit findings.
- Record non-conformities.
- Prepare recommendations.
- Prepare the audit report.
- Submit the report to management.

5. Corrective Actions

If any non-conformities are identified during the audit, the organisation must take corrective actions to resolve them. Corrective actions involve identifying the cause of the problem, implementing suitable solutions, and updating procedures to prevent similar issues in the future.

Effective corrective actions help improve software quality and ensure compliance with established standards.

Activities Performed During Corrective Actions

- Identify the root cause of the problem.
- Prepare a corrective action plan.
- Implement corrective measures.
- Update relevant procedures and documents.
- Verify the effectiveness of the corrective actions.

6. Audit Follow-up

The audit follow-up phase is the final stage of the audit process. During this phase, auditors verify whether the recommended corrective actions have been implemented successfully. They review updated records and confirm that the identified non-conformities have been resolved.

Audit follow-up supports continuous improvement by ensuring that corrective actions are effective and that similar problems do not occur again.

Activities Performed During Audit Follow-up

- Review the implemented corrective actions.
- Verify the resolution of non-conformities.
- Examine updated records and documents.
- Conduct a follow-up audit, if required.
- Close the audit.

SGOU

3.2.3 Audit Planning

In the previous section, you learned that audit planning is one of the important phases of the audit process. Although it was introduced briefly as a phase, it deserves a more detailed discussion because the success of an audit largely depends on effective planning. A well-planned audit helps auditors conduct the audit in a systematic and efficient manner while ensuring that all important aspects of the software project are evaluated.

Audit planning is carried out before the actual audit begins. During this stage, the audit team determines the objectives, scope, schedule, audit criteria, resources, and methods required for the audit. Proper planning helps avoid confusion, minimises delays, and ensures that the audit is completed within the specified time.

The following sections discuss the meaning, objectives, importance, activities, audit plan, and audit checklist in detail.

3.2.3.1 What is Audit Planning?

Audit planning is the process of preparing all the necessary activities before conducting an audit. It involves deciding what will be audited, why the audit is required, who will perform the audit, when the audit will take place, and how the audit will be conducted.

The primary purpose of audit planning is to ensure that the audit is carried out in a systematic, organised, and efficient manner. During this phase, the auditors gather preliminary information about the software project, identify the standards and procedures to be evaluated, prepare the audit schedule, and develop an audit checklist. Proper planning helps auditors focus on the most important areas and ensures that sufficient evidence is collected during the audit.

3.2.3.2 Objectives of Audit Planning

The primary objective of audit planning is to ensure that the audit is conducted in a systematic, organised, and efficient manner. Proper planning helps auditors prepare for the audit, use available resources effectively, and complete the audit within the specified time. It also ensures that the audit focuses on the most important areas and produces reliable results.

The main objectives of audit planning are:

- To define the objectives of the audit.
- To determine the scope of the audit.
- To identify the audit criteria and applicable standards.
- To prepare the audit schedule.
- To allocate resources and assign responsibilities.

SGOU

- To ensure that all important areas are covered during the audit.
- To improve the efficiency and effectiveness of the audit process.

Proper audit planning provides a clear direction for the audit, improves coordination among the audit team, reduces the chances of overlooking important activities, and helps produce accurate and reliable audit findings.

3.2.3.3 Activities in Audit Planning

Audit planning consists of a number of activities that help auditors prepare for the audit in a systematic manner. These activities ensure that the audit is properly organised, all necessary resources are available, and the audit objectives can be achieved effectively. By completing these activities before the actual audit, auditors can conduct the audit smoothly and efficiently. The major activities involved in audit planning are shown in Figure 3.2.2 and are explained below.



Fig. 3.2.2 Steps in Audit Planning

1. Defining Audit Objectives

The first activity in audit planning is to define the objectives of the audit. The audit objectives specify the purpose of the audit and what the auditors intend to achieve. Clearly defined objectives help the audit team focus on the important areas of the software project.

Example: The objective of an audit may be to verify whether the software development process complies with the organisation's quality standards.

2. Determining the Audit Scope

The audit scope defines the boundaries of the audit. It specifies the project, department, process, or system that will be examined. A clearly defined scope helps auditors concentrate on relevant activities and avoids unnecessary work.

SGOU

Example: The scope of an audit may include software requirements analysis, system design, coding, testing, and documentation.

3. Identifying Audit Criteria

Audit criteria are the standards, policies, or requirements used to evaluate the software project. During planning, auditors identify the criteria against which the collected evidence will be compared.

Common audit criteria include:

- Organisational policies
- Software development procedures
- ISO 9001 requirements
- Customer requirements
- Legal and regulatory requirements

4. Selecting the Audit Team

A suitable audit team is selected based on the nature and complexity of the audit. The team usually consists of qualified auditors with knowledge of software development and quality management. One auditor is generally appointed as the lead auditor to coordinate the audit activities.

Example: A software company may appoint one lead auditor and two supporting auditors to conduct the audit of a large software project.

5. Preparing the Audit Schedule

An audit schedule specifies the dates, time, duration, and sequence of audit activities. A well-prepared schedule helps the audit team complete the audit within the planned time and minimises disruption to the organisation's regular work.

Example: The audit may be scheduled for two days, with the first day allocated for document review and the second day for interviews and observations.

6. Identifying Required Resources

Before the audit begins, the auditors identify all the resources required for conducting the audit effectively. These resources may include project documents, computers, meeting rooms, internet access, and audit forms. Ensuring the availability of these resources helps the audit proceed without unnecessary interruptions.

7. Preparing the Audit Checklist

The final activity in audit planning is the preparation of an audit checklist. The checklist contains the questions and verification points that guide the auditors during the audit. It helps ensure that all important aspects of the software project are examined consistently.

SGOU

Example: An audit checklist may include questions such as:

- Are software requirements properly documented?
- Are coding standards being followed?
- Are test reports available?
- Is user documentation complete?

3.2.4 Audit Execution

In the previous section, you learned about audit planning, where the audit objectives, scope, schedule, resources, and checklist are prepared. Once all planning activities are completed, the audit moves to the next stage, known as audit execution. This is the most important phase of the audit process because it involves carrying out the planned audit activities, collecting evidence, evaluating compliance, and recording audit findings.

During audit execution, auditors verify whether the software development activities, processes, and quality management practices comply with the specified standards and organisational requirements. The evidence collected during this phase forms the basis for the audit report and helps the organisation identify areas for improvement.

The following sections explain the concept, objectives, activities, audit evidence, audit findings, and audit report in detail.

3.2.4.1 What is Audit Execution?

Audit execution is the phase in which the audit plan is implemented, and the actual audit is conducted. During this phase, auditors examine documents, observe software development activities, interview project personnel, and collect evidence to determine whether the organisation complies with the specified standards and procedures.

The primary purpose of audit execution is to gather sufficient and reliable evidence that supports the audit findings. The collected evidence is compared with the audit criteria to identify any non-conformities, observations, or opportunities for improvement.

3.2.4.2 Objectives of Audit Execution

The primary objective of audit execution is to evaluate whether the software development activities, processes, and products comply with the specified standards and organisational requirements. During this phase, auditors collect evidence, assess compliance, identify areas for improvement, and document the audit findings.

The main objectives of audit execution are to:

- verify compliance with specified standards and procedures.
- collect sufficient and reliable audit evidence.
- evaluate software development activities.

SGOU

- identify non-conformities and weaknesses.
- record audit findings accurately.
- support continuous improvement.

3.2.4.3 Activities in Audit Execution

Audit execution involves several activities that help auditors systematically evaluate software processes and products. They are given below:



Fig. 3.2.3 Activities in Audit Execution

1. Opening Meeting

The audit begins with an opening meeting between the audit team and the representatives of the organisation. During this meeting, the auditors explain the audit objectives, scope, schedule, and methods to be followed.

2. Document Review

The auditors examine project documents such as software requirements, design documents, coding standards, test reports, quality records, and previous audit reports.

3. Observation of Activities

The auditors observe software development activities to verify whether the approved procedures are being followed during project execution.

4. Interviewing Personnel

The auditors interact with project managers, developers, testers, and quality assurance personnel to understand how software development activities are performed.

SGOU

5. Collecting Audit Evidence

The auditors collect evidence through document review, interviews, observations, and examination of project records. The evidence collected should be sufficient, relevant, and reliable.

6. Recording Audit Findings

The collected evidence is compared with the audit criteria. Based on this comparison, the auditors record observations, non-conformities, and opportunities for improvement.

7. Closing Meeting

The audit concludes with a closing meeting in which the auditors present the major findings and discuss the next steps with the organisation's management.

3.2.5 ISO 9001 Overview

In the previous sections, you learned about the audit process, audit planning, and audit execution. During an audit, software development activities, processes, and products are evaluated to determine whether they comply with established standards and organisational requirements. However, for an audit to be effective, there must be a recognised standard against which the organisation can be evaluated. One of the most widely accepted international standards used for this purpose is ISO 9001.

The International Organisation for Standardisation (ISO) is an independent, non-governmental international organisation that develops and publishes international standards for products, services, and management systems. Established in 1947 and headquartered in Geneva, Switzerland, ISO brings together experts from member countries to develop standards that promote quality, safety, efficiency, and consistency across various industries.

Among the many standards developed by ISO, ISO 9001 is the internationally recognised standard for Quality Management Systems (QMS). It provides a framework for establishing, implementing, maintaining, and continually improving a quality management system. The standard helps organisations consistently deliver products and services that meet customer expectations as well as statutory and regulatory requirements.

Unlike technical standards that specify product characteristics, ISO 9001 focuses on how an organisation manages its processes to achieve consistent quality. It encourages organisations to adopt a process-based approach, maintain proper documentation, monitor performance, manage risks, and continually improve their operations.

3.2.6 ISO 9001 Certification

In the previous section, you learned about ISO 9001 and its role in establishing an effective Quality Management System (QMS). Organisations that implement ISO 9001 often seek formal recognition that their quality management system meets the requirements of the standard. This recognition is known as ISO 9001 certification.

SGOU

ISO 9001 certification is the process through which an independent certification body verifies that an organisation's Quality Management System complies with the requirements of the ISO 9001 standard. It demonstrates the organisation's commitment to delivering consistent quality products and services while meeting customer and regulatory requirements.

The certification process includes evaluating the quality management system, reviewing documents, verifying process implementation, and assessing compliance with ISO 9001 requirements. Organisations that successfully meet these requirements are awarded the ISO 9001 certificate.

ISO 9001 certification helps organisations improve customer confidence, enhance product and service quality, strengthen their reputation, and support continual improvement. The following sections discuss the objectives, certification process, required documents, and benefits of ISO 9001 certification.

3.2.6.1 ISO 9001 Certification Process

ISO 9001 certification is a systematic process used to verify whether an organisation's Quality Management System (QMS) complies with the requirements of the ISO 9001 standard. The certification process involves a series of well-defined steps, beginning with an assessment of the existing quality management practices and ending with the award of the ISO 9001 certificate. Each step helps ensure that the organisation has established, implemented, and maintained an effective Quality Management System.

The certification process not only evaluates compliance with ISO 9001 requirements but also encourages organisations to improve their processes, enhance customer satisfaction, and achieve continual improvement. Organisations that successfully complete all the stages of the certification process are granted ISO 9001 certification by an accredited certification body.

The major steps involved in the ISO 9001 certification process are illustrated in Figure 3.2.4 and are explained in the following sections.

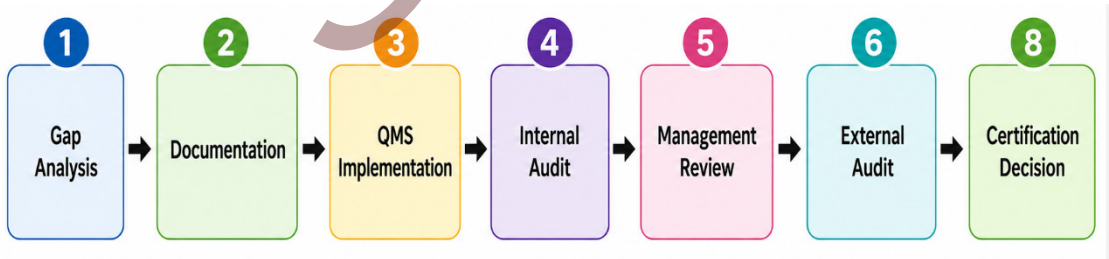


Fig. 3.2.4 ISO 9001 Certification Process

1. Gap Analysis

The first step is to assess the organisation's existing quality management practices and identify gaps between the current system and the requirements of ISO 9001. This helps determine the improvements needed before certification.

SGOU

2. Documentation

The organisation prepares the documents required for the Quality Management System, such as quality policies, objectives, procedures, work instructions, and records. Proper documentation ensures that all processes are clearly defined and consistently followed.

3. QMS Implementation

The documented Quality Management System is implemented throughout the organisation. Employees are trained to follow the established procedures, and the organisation begins operating according to the requirements of ISO 9001.

4. Internal Audit

An internal audit is conducted to verify whether the implemented Quality Management System is functioning effectively and complies with ISO 9001 requirements. Any non-conformities identified during the audit are recorded for corrective action.

5. Management Review

The organisation's top management reviews the performance of the Quality Management System. The review evaluates audit findings, customer feedback, process performance, and opportunities for improvement to ensure that the system remains effective.

6. External Audit

An accredited certification body conducts an external audit to assess whether the organisation's Quality Management System complies with the requirements of ISO 9001. The auditors review documents, observe activities, and interview employees before making their assessment.

7. Certification Decision

Based on the results of the external audit, the certification body decides whether the organisation satisfies the requirements of ISO 9001. If all requirements are met, the organisation is awarded the ISO 9001 certificate.

8. Surveillance Audit

After certification, periodic surveillance audits are conducted to ensure that the organisation continues to comply with ISO 9001 requirements and maintains its Quality Management System effectively. These audits support continual improvement and help maintain the validity of the certification.

SGOU

Recap

- The audit process is a systematic sequence of activities used to evaluate software processes, products, and quality management systems.
- The audit process consists of seven phases, from audit initiation to audit follow-up.
- Audit initiation defines the audit objectives, scope, criteria, and audit team.
- Audit preparation involves reviewing project documents, policies, and previous audit reports before the audit begins.
- Audit execution includes collecting evidence, reviewing documents, observing activities, and interviewing project personnel.
- Audit reporting summarises the findings, non-conformities, and recommendations for improvement.
- Corrective actions help resolve non-conformities and prevent similar problems in the future.
- Audit follow-up verifies whether the corrective actions have been successfully implemented.
- Software audits include internal, external, product, process, system, compliance, quality, and security audits.
- Audit planning defines the objectives, scope, criteria, schedule, resources, and audit checklist.
- Audit execution verifies compliance, collects evidence, and records audit findings systematically.
- ISO 9001 is an international standard for establishing and improving a Quality Management System (QMS).
- The ISO 9001 certification process includes gap analysis, documentation, QMS implementation, audits, management review, certification, and surveillance.

SGOU

Objective Type Questions

1. What is the first phase of the audit process?
2. Which phase follows audit reporting?
3. What is the final phase of the audit process?
4. What is collected during audit execution?
5. What are failures to meet specified requirements called?
6. Which international standard specifies requirements for a Quality Management System?
7. Which phase prepares the audit checklist?

Answers to Objective Type Questions

1. Audit Initiation.
2. Corrective Actions.
3. Audit Follow-up.
4. Evidence.
5. Non-conformities.
6. ISO 9001.
7. Audit Planning.

Assignments

1. Explain the audit process and describe its seven phases.
2. Describe the objectives and major activities involved in audit planning.
3. Explain the objectives and activities of audit execution with suitable examples.
4. What is ISO 9001? Explain its importance in maintaining software quality.
5. Discuss the major steps involved in the ISO 9001 certification process.

SGOU

Suggested Reading

1. Hoyle, David. *ISO 9000 Quality Systems Handbook*. 7th ed., Routledge.

2. O'Regan, Gerard. *Introduction to Software Quality*. Springer, 2014.

3. International Organization for Standardization. *ISO/IEC 29110-1-1:2024: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 1-1: Overview*. ISO, 2024.

4. International Organization for Standardization. *ISO/IEC 29110-5-1-1:2025: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 5-1-1: Software Engineering Guidelines for the Generic Entry Profile*. ISO, 2025.

Reference

1. Galin, Daniel. *Software Quality: Concepts and Practice*. 2nd ed., Wiley-IEEE Computer Society Press, 2018.

2. O'Regan, Gerard. *Introduction to Software Process Improvement*. Springer, 2011.

3. International Organization for Standardization. *ISO 9001:2015: Quality Management Systems—Requirements*. ISO, 2015.

SGOU



Audit According to the CMMI

Learning Outcomes

After completion of this unit, learner will be able to:

- describe the concept of Capability Maturity Model Integration (CMMI)
- explain the CMMI maturity levels
- describe the SCAMPI assessment method and its objectives
- explain the stages of the SCAMPI assessment process

Prerequisite

Software organisations follow a systematic process to develop high-quality software. You have already learned about the software development life cycle, quality management, software quality audits, and the ISO 9001 standard for Quality Management Systems (QMS). You also know that organisations must follow defined processes and quality standards to produce reliable software products.

Consider a software company developing a Hospital Management System. The company has established coding standards, testing procedures, and documentation practices to ensure software quality. However, simply defining these processes is not enough. The organisation also needs to determine how effectively these processes are implemented, identify areas that require improvement, and measure its overall process capability. This requires a structured process improvement framework and a systematic assessment method.

This unit introduces the Capability Maturity Model Integration (CMMI), which helps organisations assess and improve their software development processes. It also explains the SCAMPI assessment method, which is used to evaluate process maturity based on the CMMI framework. In addition, you will learn why software audits are important for Very Small Entities (VSEs) and how a simple audit approach helps them improve software quality and achieve continuous process improvement.

SGOU

Keywords

Capability Maturity Model Integration (CMMI), CMMI Maturity Levels, SCAMPI Assessment Method, Very Small Entities (VSEs)

Discussion

3.3.1 Audit According to the CMMI

In the previous unit, you learned about the audit process and the ISO 9001 standard for Quality Management Systems (QMS). ISO 9001 helps organisations establish standardised processes and consistently deliver quality products and services. However, software organisations also need to evaluate how effectively these processes are implemented and identify opportunities for continual improvement.

To support this objective, many organisations use the Capability Maturity Model Integration (CMMI), a widely accepted framework for assessing and improving software development processes. While ISO 9001 focuses on quality management, CMMI focuses on measuring the capability and maturity of organisational processes. Together, they help organisations improve software quality, increase process efficiency, and enhance customer satisfaction.

One of the most widely used methods for evaluating software processes based on CMMI is the Standard CMMI Appraisal Method for Process Improvement (SCAMPI). It provides a structured approach for assessing process capability, identifying strengths and weaknesses, and supporting continual process improvement.

This unit introduces the concepts of CMMI, its maturity levels, the SCAMPI assessment method, and the audit approach followed in Very Small Entities (VSEs).

3.3.2 Capability Maturity Model Integration (CMMI)

The Capability Maturity Model Integration (CMMI) is a globally recognised framework that helps organisations improve the quality and efficiency of their business and software development processes. It provides a structured approach for evaluating existing processes, identifying areas for improvement, and implementing best practices to achieve better organisational performance.

CMMI was originally developed by the Software Engineering Institute (SEI) at Carnegie Mellon University to help organisations improve software development processes. Today, it is widely used in software, systems engineering, product development, and service organisations to improve process capability, reduce project risks, and deliver high-quality products and services.

Unlike ISO 9001, which specifies the requirements for a Quality Management System, CMMI focuses on how well an organisation's processes are defined, managed, measured, and continuously improved. It provides a roadmap that enables organisations to move from informal and unpredictable processes to well-defined, controlled, and continuously

SGOU

improving processes.

By implementing CMMI, organisations can standardise software development activities, improve project management, enhance product quality, reduce development costs, and increase customer satisfaction. It also helps organisations assess their process maturity and identify opportunities for continual improvement.

3.3.2.1 CMMI Maturity Levels

One of the key features of the Capability Maturity Model Integration (CMMI) is its maturity level model, which helps organisations assess the capability and effectiveness of their software development processes. A maturity level indicates how well an organisation's processes are defined, managed, measured, and continuously improved.

The CMMI maturity model consists of five maturity levels. Each level represents a stage of process improvement and provides a roadmap for organisations to achieve higher levels of process capability. As organisations progress through these levels, their software development processes become more organised, predictable, and efficient, resulting in improved product quality and customer satisfaction.

The five maturity levels of CMMI are illustrated in Figure 3.3.1 and are explained in the following sections.

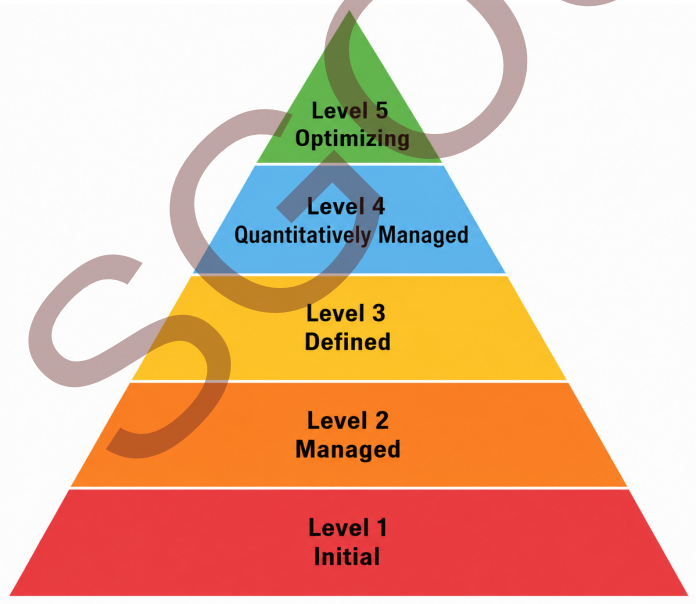


Fig. 3.3.1 CMMI Maturity Levels

1. Level 1 – Initial

At Level 1, software development processes are informal and unpredictable. Organisations do not follow standardised procedures, and project success largely depends on the skills and experience of individual team members. As a result, projects may face frequent changes, higher risks, delays, cost overruns, and inconsistent software quality.

SGOU

2. Level 2 – Managed

At Level 2, organisations establish basic project management practices to plan, monitor, and control software projects. Requirements, schedules, and resources are managed systematically, resulting in improved project control and more consistent software development activities.

3. Level 3 – Defined

At Level 3, software development processes are well documented, standardised, and followed consistently across the organisation. Common procedures and guidelines improve communication, coordination, and the overall quality of software development.

4. Level 4 – Quantitatively Managed

At Level 4, organisations measure and control their software processes using quantitative data and performance metrics. Decisions are based on analysis of process performance, leading to better quality control and more predictable project outcomes.

5. Level 5 – Optimising

At Level 5, organisations focus on continuous process improvement by identifying weaknesses, adopting innovative practices, and preventing problems before they occur. This results in highly efficient processes, improved software quality, and greater customer satisfaction.

3.3.3 SCAMPI Assessment Method

In the previous section, you learned about the Capability Maturity Model Integration (CMMI), which helps organisations improve their software development processes. However, organisations also need a method to check whether these processes are being followed properly. For this purpose, the Standard CMMI Appraisal Method for Process Improvement (SCAMPI) is used.

SCAMPI is an appraisal method used to evaluate an organisation's software development processes based on the CMMI framework. It helps determine how well the organisation's processes are implemented and identifies areas that need improvement.

During a SCAMPI assessment, auditors collect information by reviewing documents, interviewing employees, observing activities, and examining project records. Based on this information, they evaluate the organisation's process maturity and provide suggestions for improvement.

SCAMPI helps organisations improve software quality, strengthen their development processes, and support continuous improvement.

3.3.3.1 Objectives of SCAMPI

The primary objective of SCAMPI (Standard CMMI Appraisal Method for Process Improvement) is to evaluate an organisation's software development processes based on the CMMI framework. It helps organisations understand the effectiveness of their

SGOU

processes, identify areas for improvement, and enhance overall process capability.

The main objectives of SCAMPI are to:

- assess the capability of organisational processes.
- determine the maturity level of the organisation.
- identify the strengths and weaknesses of existing processes.
- verify compliance with CMMI practices.
- support continuous process improvement.
- improve software quality and organisational performance.

3.3.3.2 SCAMPI Assessment Process

The SCAMPI Assessment Process is a systematic sequence of activities used to evaluate an organisation's software development processes based on the CMMI framework. It provides a structured approach for collecting evidence, assessing process capability, identifying strengths and weaknesses, and recommending improvements.

A well-defined assessment process ensures that the appraisal is conducted in a consistent, objective, and reliable manner. The findings obtained from the assessment help organisations understand their current process maturity and identify opportunities for continual improvement.

The SCAMPI assessment process generally consists of the following six stages:

- Assessment Planning
- Assessment Preparation
- Data Collection
- Evidence Verification
- Process Evaluation
- Reporting the Results

These stages are illustrated in Figure 3.3.2 and are explained in the following sections.

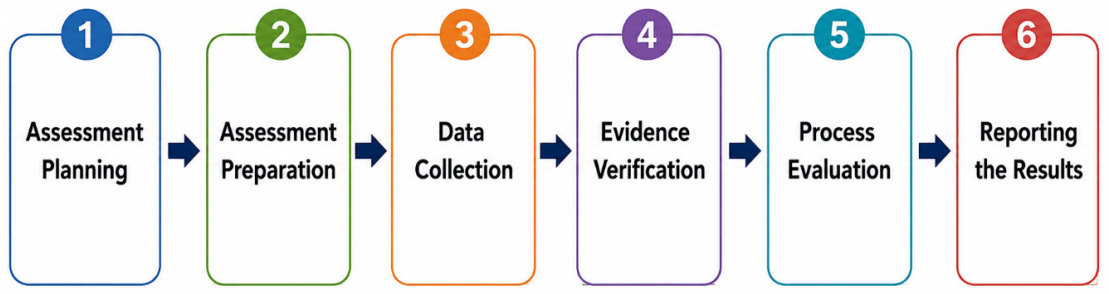


Fig. 3.3.2 SCAMPI Assessment Process

SGOU

1. **Assessment Planning:** Assessment planning is the first stage of the SCAMPI assessment process. During this stage, the objectives, scope, schedule, and assessment team are decided. The organisation also identifies the projects and process areas that will be assessed. Proper planning helps ensure that the assessment is carried out smoothly.

2. **Assessment Preparation:** During the preparation stage, the assessment team reviews project documents, process documents, and previous assessment reports. Necessary resources are arranged, and employees are informed about the assessment activities. This helps the team understand the organisation's processes before the assessment begins.

3. **Data Collection:** In this stage, the assessment team collects evidence from various sources such as project documents, interviews with employees, observations of work activities, and project records. The collected information is used to evaluate whether the organisation's processes follow CMMI practices.

4. **Evidence Verification:** The collected evidence is carefully examined to ensure that it is accurate, complete, and reliable. If additional information is required, the assessment team collects further evidence before proceeding with the evaluation.

5. **Process Evaluation:** The verified evidence is compared with the CMMI requirements to determine how effectively the organisation's processes are implemented. The assessment team identifies strengths, weaknesses, and areas that require improvement.

6. **Reporting the Results:** The final stage is to prepare and present the assessment report. The report summarises the assessment findings, process maturity, strengths, weaknesses, and recommendations for improvement. The organisation uses this report to plan future process improvement activities.

3.3.4 Audits for Very Small Entities (VSEs)

In the previous section, you learned about the SCAMPI assessment method, which is used to evaluate software development processes based on the CMMI framework. While SCAMPI is suitable for many organisations, its implementation may be difficult for organisations with limited staff, budget, and resources. Such organisations require a simpler and more practical approach to software auditing.

A Very Small Entity (VSE) is an organisation, department, or project that typically has up to 25 employees. Most software startups, small software development firms, freelance development teams, and IT service providers fall into this category. These organisations usually work on small projects with limited resources but still need to deliver high-quality software to their customers.

Although VSEs are small in size, they must ensure that their software development processes are well organised and produce reliable software products. Regular audits help these organisations identify weaknesses, improve software quality, and maintain customer confidence. Therefore, audit practices for VSEs are designed to be simple, practical, and easy to implement.

SGOU

3.3.4.1 Need for Audits in Very Small Entities

Software quality is important for organisations of every size. Even though Very Small Entities have fewer employees and simpler projects, they still need to ensure that their software meets customer requirements and follows good development practices. Regular audits help VSEs evaluate their processes, identify problems at an early stage, and improve the overall quality of their products.

Audits also help VSEs use their limited resources effectively. By identifying process weaknesses and recommending improvements, audits reduce errors, improve project management, and increase customer satisfaction.

The major reasons for conducting audits in VSEs are:

- improve software quality.
- identify process weaknesses.
- ensure compliance with organisational procedures.
- improve customer confidence.
- reduce project risks.
- support continuous process improvement.

3.3.4.2 Challenges in Conducting Audits for Very Small Entities

Conducting audits in Very Small Entities can be difficult because these organisations often operate with limited resources. Employees may perform multiple responsibilities, and there may not be a separate quality assurance or audit team. As a result, audits must be carefully planned and simplified.

Some common challenges faced by VSEs are:

7. **Limited Staff :** Employees often perform multiple roles, leaving little time for audit activities.
8. **Limited Budget :** Financial resources for quality improvement and audits are often limited.
9. **Limited Documentation :** Small organisations may maintain only essential project documents.
10. **Lack of Audit Expertise:** Employees may have limited knowledge of software auditing.
11. **Time Constraints :** Tight project schedules reduce the time available for audits.
12. **Resource Constraints:** Limited tools and infrastructure make auditing more challenging.

SGOU

3.3.4.3 Audit Approach for Very Small Entities

Audits in Very Small Entities should be practical, simple, and focused on improving software quality without creating unnecessary documentation. The audit approach should match the size and complexity of the organisation while ensuring that important software development activities are properly evaluated.

The audit process generally includes the following steps:

- Define the audit objectives.
- Plan the audit.
- Review important project documents.
- Interview team members and observe activities.
- Record audit findings.
- Recommend corrective actions.
- Conduct follow-up activities.



Fig. 3.3.3 Audit Approach for Very Small Entities

1. Define Audit Objectives

The first step is to clearly identify the purpose of the audit. The organisation determines what needs to be evaluated, such as a software project, development process, or quality management activity. Well-defined objectives help the audit team focus on the important areas during the assessment.

2. Plan the Audit

After defining the objectives, a detailed audit plan is prepared. This includes deciding the audit schedule, selecting the audit team, and identifying the documents and resources required. Proper planning ensures that the audit is conducted in an organised and efficient manner.

3. Review Documents

The audit team reviews important project documents before conducting the assessment. These may include requirement specifications, design documents, project plans, test

SGOU

reports, and quality records. Reviewing these documents helps auditors understand the project and verify whether standard procedures are being followed.

4. Conduct Interviews and Observations

The auditors interact with project team members to understand how software development activities are carried out. They also observe work practices and project activities to verify whether the documented processes are followed in actual practice. This provides valuable information for the assessment.

5. Record Findings

During the assessment, the audit team records all important observations and findings. These include strengths, weaknesses, and any non-conformities identified in the software development process. Proper documentation of findings provides the basis for the final audit report.

6. Recommend Corrective Actions

Based on the audit findings, the auditors suggest suitable corrective actions to address the identified issues. These recommendations help improve software quality, strengthen development processes, and prevent similar problems from occurring in the future.

7. Follow Up

The final step is to verify whether the recommended corrective actions have been successfully implemented. The auditors review the updated processes and documents to ensure that the identified issues have been resolved. This helps organisations achieve continuous improvement and maintain software quality.

Recap

- Capability Maturity Model Integration (CMMI) is a framework that helps organisations improve their software development processes.
- CMMI uses five maturity levels to measure and improve process capability and software quality.
- SCAMPI is an appraisal method that evaluates software development processes using the CMMI framework.
- The main objectives of SCAMPI are to assess process capability, determine maturity levels, and support continuous process improvement.
- The SCAMPI assessment process includes planning, preparation, data collection, evidence verification, process evaluation, and reporting.
- A Very Small Entity (VSE) is an organisation with up to 25 employees and limited resources.

SGOU

- Regular audits help VSEs improve software quality, identify process weaknesses, and reduce project risks.
- VSEs face audit challenges such as limited staff, budget, documentation, expertise, time, and resources.
- The audit approach for VSEs includes planning the audit, reviewing documents, recording findings, recommending corrective actions, and conducting follow-up activities.
- A systematic audit approach helps organisations achieve continuous process improvement and maintain software quality.

Objective Type Questions

1. What does CMMI stand for?
2. Which framework improves software development processes?
3. Which organisation originally developed CMMI?
4. How many maturity levels are defined in CMMI?
5. Which CMMI level represents informal and unpredictable processes?
6. Which CMMI level establishes basic project management practices?
7. What does SCAMPI stand for?
8. Which method evaluates processes based on the CMMI framework?
9. Which type of organisation is abbreviated as VSE?
10. Which activity verifies the implementation of corrective actions?

SGOU

Answers to Objective Type Questions

1. Capability Maturity Model Integration.
2. CMMI.
3. SEI.
4. Five.
5. Initial.
6. Managed.
7. Standard CMMI Appraisal Method for Process Improvement.
8. SCAMPI.
9. Very Small Entity.
10. Follow-up.

Assignments

1. Explain the Capability Maturity Model Integration (CMMI) framework and describe its five maturity levels.
2. Describe the SCAMPI assessment method and explain its objectives in software process improvement.
3. Discuss the stages involved in the SCAMPI assessment process with suitable explanations.
4. Explain the need for audits in Very Small Entities (VSEs) and discuss the challenges faced during software audits.
5. Describe the audit approach followed in Very Small Entities (VSEs) with the steps involved.

SGOU

Suggested Reading

1. Hoyle, David. *ISO 9000 Quality Systems Handbook*. 7th ed., Routledge.

2. O'Regan, Gerard. *Introduction to Software Quality*. Springer, 2014.

3. International Organization for Standardization. *ISO/IEC 29110-1-1:2024: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 1-1: Overview*. ISO, 2024.

4. International Organization for Standardization. *ISO/IEC 29110-5-1-1:2025: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 5-1-1: Software Engineering Guidelines for the Generic Entry Profile*. ISO, 2025.

Reference

1. Galin, Daniel. *Software Quality: Concepts and Practice*. 2nd ed., Wiley-IEEE Computer Society Press, 2018.

2. O'Regan, Gerard. *Introduction to Software Process Improvement*. Springer, 2011.

3. International Organization for Standardization. *ISO 9001:2015: Quality Management Systems—Requirements*. ISO, 2015.

SGOU



Software Quality Auditing

Learning Outcomes

After completion of this unit, learner will be able to:

- describe the concept of Software Quality Auditing
- describe the concept of Software Quality Assurance (SQA)
- explain the major activities of Software Quality Assurance
- discuss the importance and components of an SQA Plan
- explain the audit activities involved in an SQA Plan

Prerequisite

Software is used in many areas of daily life, such as online banking, e-commerce, hospital management, online learning, and examination systems. You already know that software is developed through activities such as requirements analysis, design, coding, testing, and documentation. You also understand that software should work correctly, satisfy user requirements, and provide reliable services.

For example, consider an Online Examination System used by a university. The system allows students to register, attend examinations, submit answers, and view results. Even if the system appears to work correctly, it is important to ensure that the development team followed the required standards and procedures throughout the project. Were the requirements reviewed? Was the code properly tested? Were defects recorded and corrected? Were project documents maintained correctly?

To answer these questions, organisations follow systematic quality practices throughout the software development process. This unit introduces Software Quality Assurance (SQA) and Software Quality Auditing, which help ensure that software development activities comply with defined standards and continuously improve software quality. You will also learn about the SQA Plan, audit activities, and a practical case study that demonstrates how software quality is maintained in real-world projects.

SGOU

Keywords

Software Quality Assurance (SQA), SQA Plan, Audit Activities, Software Quality Improvement, Quality Standards

Discussion

In the previous unit, you learned about software process assessment using CMMI, the SCAMPI assessment method, and audit practices for Very Small Entities (VSEs). These approaches help organisations evaluate and improve their software development processes. However, the ultimate goal of every audit is to ensure that the software being developed meets the required quality standards.

Software quality is one of the most important factors in software development. High-quality software is reliable, efficient, secure, and satisfies user requirements. To achieve this, organisations conduct software quality audits, which systematically evaluate software processes, products, and quality assurance activities. These audits help identify weaknesses, verify compliance with standards, and recommend improvements that lead to better software quality.

This unit explains the concept of software quality auditing, its relationship with the Software Quality Assurance (SQA) Plan, and the presentation of an audit case study.

3.4.1 What is Software Quality Auditing?

Software Quality Auditing is a systematic and independent examination of software development processes, activities, documents, and quality records to determine whether they comply with specified standards, procedures, and requirements.

A software quality audit does not focus only on finding defects in the final software product. Instead, it examines how the software is developed and managed. The audit may cover activities such as requirements analysis, software design, coding, testing, configuration management, documentation, and quality assurance.

During an audit, auditors collect evidence through document reviews, interviews, observations, and examination of project records. The collected evidence is compared with the defined audit criteria. Any differences or failures to meet the requirements are recorded as findings or non-conformities.

For example, suppose a software company has a procedure requiring all software requirements to be reviewed and approved before development begins. During an audit, the auditor examines the requirement documents and approval records.

If some requirements do not have evidence of review or approval, the auditor may report this as a non-conformity. The organisation can then take corrective action to improve its requirements management process. Thus, software quality auditing helps organisations check whether their planned quality practices are actually followed in practice.

SGOU

3.4.2 Objectives of Software Quality Auditing

The main objectives of software quality auditing are to:

1. **Verify compliance:** Check whether software development activities follow specified standards, procedures, and requirements.

2. **Evaluate software processes:** Examine whether important software development activities are properly performed.

3. **Identify non-conformities:** Find activities or documents that do not meet the specified requirements.

4. **Improve software quality:** Identify weaknesses and suggest improvements to development and quality processes.

5. **Verify documentation:** Ensure that important project documents and quality records are properly maintained.

6. **Support continuous improvement:** Provide information that can be used to improve processes and prevent similar problems in the future.

3.4.3 Software Quality Assurance (SQA)

Software quality is an important concern throughout the software development life cycle. Developing software that satisfies functional requirements alone is not sufficient. The software should also be reliable, maintainable, secure, efficient, and consistent with the specified quality requirements. Therefore, organisations follow systematic activities to plan, monitor, and improve software quality.

Software Quality Assurance (SQA) is a set of planned and systematic activities performed throughout the software development life cycle to ensure that software processes and products meet specified quality requirements.

SQA focuses on preventing quality problems and improving software processes, rather than identifying problems only after the software has been completed. It ensures that appropriate standards, procedures, methods, and quality practices are defined and followed during software development.

3.4.3.1 Need for Software Quality Assurance

Software projects involve several activities such as requirements analysis, design, coding, testing, documentation, configuration management, and maintenance. If these activities are not properly controlled, they may lead to defects, delays, increased development costs, and customer dissatisfaction.

SQA helps organisations establish systematic quality practices from the beginning of the project. It ensures that quality requirements are considered at every stage of development.

SGOU

The major needs of SQA are to:

- Ensure that software processes follow defined standards and procedures.
- Identify and prevent quality problems at an early stage.
- Improve the reliability and maintainability of software.
- Reduce software defects and development risks.
- Ensure consistent quality throughout the development process.
- Support continuous improvement of software processes.
- Increase customer confidence and satisfaction.

3.4.3.2 Major Activities of Software Quality Assurance

Software Quality Assurance (SQA) involves a set of planned activities that help ensure the quality of software processes and products throughout the software development life cycle. These activities help organisations follow defined standards, identify problems at an early stage, and continuously improve their software development practices.

The major activities of SQA are explained below.

1. Defining Quality Standards

SQA involves identifying and establishing the standards, guidelines, and procedures that should be followed during software development. These standards help ensure consistency in development activities and software quality.

Quality standards may include coding standards, documentation standards, testing standards, review procedures, and organisational quality policies.

For example, an organisation may define a standard format for writing software requirements and specify coding rules that all developers should follow.

2. Preparing Quality Plans

7. Quality planning determines how software quality will be managed throughout a project. The SQA Plan specifies the quality objectives, applicable standards, responsibilities, reviews, audits, testing activities, documentation requirements, and other quality-related activities.

A well-prepared SQA Plan provides a clear framework for carrying out quality activities in a systematic manner.

For example, an SQA Plan may specify that requirements must be reviewed before coding and that software testing must be completed before the final release.

3. Conducting Reviews

A review is a systematic examination of a software work product to identify errors, omissions, and areas for improvement at an early stage. Reviews can be conducted on

SGOU

requirements, design documents, source code, test plans, and other project documents. Reviews help identify problems before they become costly to correct.

For example, before software development begins, the requirements document can be reviewed to identify unclear, incomplete, or conflicting requirements.

4. Conducting Audits

An audit is a systematic examination of software processes, activities, documents, and records to determine whether they comply with specified standards, procedures, and requirements.

Audits provide objective evidence about whether the quality practices defined by the organisation are being followed.

For example, if the SQA Plan requires code reviews for all software modules, an audit can examine code review records to verify whether the required reviews were conducted.

5. Performing Software Testing

Software testing is the process of evaluating software to identify defects and verify whether it satisfies specified requirements. SQA ensures that appropriate testing procedures and standards are planned and followed.

Testing activities may include preparing test cases, executing tests, recording test results, and reporting defects.

For example, In an Online Examination System, testing can verify whether students can log in, attend an examination, submit answers, and receive results correctly.

6. Managing Defects

Defect management involves identifying, recording, analysing, tracking, and resolving defects found during software development and testing.

SQA ensures that defects are properly documented and handled according to defined procedures. This helps prevent defects from being ignored or repeated.

For example, if a login error is identified during testing, it can be recorded, assigned to a developer, corrected, tested again, and closed after verification.

7. Controlling Changes

Changes to requirements, design, source code, or other project items are common during software development. SQA ensures that such changes are properly reviewed, approved, documented, and controlled.

Effective change control helps prevent unauthorised changes from affecting software quality.

For example, if a customer requests a new feature, the request should be evaluated and approved before the feature is added to the software.

SGOU

8. Maintaining Quality Records

SQA requires important quality-related information to be properly recorded and maintained. These records provide evidence that planned quality activities have been performed.

Common quality records include:

- review records
- audit reports
- test cases and test results
- defect records
- change records
- corrective action records

These records are useful for monitoring project quality, conducting audits, and supporting future process improvement.

For example, Test reports can be maintained to show which test cases were executed, which tests passed or failed, and which defects were identified.

9. Managing Risks

Software projects may face risks related to requirements, technology, security, resources, schedules, and quality. SQA helps identify and manage risks that may affect software quality.

Potential risks are analysed and suitable actions are planned to reduce their impact.

For example, if insufficient testing is identified as a quality risk, the project team may allocate additional time and resources for testing.

10. Improving Software Processes

SQA supports the continuous improvement of software development processes. Information obtained from reviews, audits, testing, defect records, and other quality activities can be analysed to identify areas that need improvement.

Process improvements help organisations prevent repeated problems and achieve better quality in future projects.

For example, If repeated defects occur because software requirements are unclear, the organisation may improve its requirements review and approval process for future projects.

3.4.4 SQA Plan

Software Quality Assurance (SQA) involves several activities such as defining quality standards, conducting reviews and audits, performing testing, managing defects, controlling changes, and maintaining quality records. To perform these activities

SGOU

systematically, a software project needs a proper plan. This plan is known as the Software Quality Assurance Plan (SQA Plan).

An SQA Plan is a document that describes the quality requirements, standards, procedures, responsibilities, and activities to be followed throughout a software project. It provides a structured approach for planning, monitoring, and controlling software quality.

The SQA Plan is prepared at the beginning of the project and may be updated when there are significant changes in project requirements, processes, or quality requirements.

3.4.4.1 Importance of an SQA Plan

An SQA Plan is important because software quality cannot be achieved by testing the final product alone. Quality needs to be considered throughout the development process. A properly prepared SQA Plan helps the project team follow consistent quality practices from requirements analysis to software maintenance.

The important benefits of an SQA Plan include:

1. **Provides clear quality guidelines:** It explains the quality standards and procedures that should be followed by the project team.

2. **Defines responsibilities:** It identifies who is responsible for reviews, audits, testing, documentation, and other quality activities.

3. **Supports systematic quality management:** It ensures that quality activities are planned and performed in an organised manner.

4. **Helps identify problems early:** Reviews, audits, and other planned activities can help detect problems before they become serious.

5. **Improves documentation:** It specifies the quality records and documents that need to be prepared and maintained.

6. **Supports auditing:** It provides the criteria against which software development activities can be reviewed and audited.

7. **Promotes continuous improvement:** Findings from reviews, audits, testing, and defect analysis can be used to improve software processes.

3.4.5 Audit Activities in the SQA Plan

Software quality audits are an important part of Software Quality Assurance (SQA). The SQA Plan specifies how audits will be planned, conducted, documented, and followed up during the software project. Audit activities help determine whether the software development processes and quality practices comply with the defined standards, procedures, and project requirements.

SGOU

The major audit activities included in an SQA Plan are explained below.

1. Define Audit Objectives

The first step is to clearly identify what the audit intends to examine. The objectives provide direction to the audit team and help determine the areas that need to be evaluated.

For example, an audit may aim to verify whether software testing activities are performed according to the procedures defined in the SQA Plan.

2. Define Audit Scope

The audit scope specifies the project activities, processes, documents, or products that will be examined. Clearly defining the scope prevents unnecessary activities and ensures that important areas are covered.

For example, an audit may be limited to requirements management, software testing, and defect management.

3. Identify Audit Criteria

Audit criteria are the standards, procedures, policies, plans, or requirements against which the software project is evaluated.

For example, the SQA Plan, organisational quality procedures, coding standards, and testing procedures may be used as audit criteria.

4. Prepare the Audit Schedule

The SQA Plan specifies when audits should be conducted. Audits may be scheduled at important stages of the software development life cycle or before major software releases.

A suitable schedule ensures that audit activities do not interfere unnecessarily with other project activities.

5. Assign Audit Responsibilities

The SQA Plan identifies the persons responsible for conducting and supporting the audit. Their roles and responsibilities should be clearly defined.

For example, an auditor may be responsible for collecting evidence and preparing the audit report, while project team members may provide the required documents and information.

6. Review Documents

Auditors examine relevant software project documents to understand whether the required quality practices have been followed.

Documents may include:

- Requirements specifications
- Design documents
- Project plans

SGOU

- Test plans and test reports
- Defect records
- Review records
- Change records

7. Collect Audit Evidence

Audit evidence is collected to determine whether the defined requirements are actually being followed. Evidence may be obtained through document examination, interviews, observations, and project records.

For example, if the SQA Plan requires code reviews, the auditor can examine code review records as evidence.

8. Evaluate Audit Findings

The collected evidence is compared with the audit criteria. The auditor determines whether the activities comply with the specified requirements and identifies any deviations or weaknesses.

The findings may indicate:

- **Conformity:** The specified requirement has been followed.
- **Non-conformity:** The specified requirement has not been followed.
- **Observation:** An issue that may require attention.
- **Opportunity for improvement:** A suggestion for improving an existing process.

9. Prepare the Audit Report

The audit findings are documented in an audit report. The report provides a clear record of the audit activities and results.

An audit report may include:

- audit objectives
- audit scope
- audit criteria
- activities examined
- evidence collected
- findings
- non-conformities
- recommendations

SGOU

10. Recommend Corrective Actions

When non-conformities are identified, suitable corrective actions are recommended. Corrective actions should address the identified problems and help prevent their recurrence.

For example, if testing records are incomplete, the organisation may introduce a standard test-result format and require testers to maintain the records properly.

11. Conduct Follow-up Activities

Follow-up activities are performed to verify whether the recommended corrective actions have been implemented successfully. The audit finding can be closed after sufficient evidence shows that the problem has been properly addressed.

3.4.6 Presentation of an Audit Case Study

Software Quality Audit of an Online Examination System

Consider a software organisation developing an Online Examination System for a university. The system allows students to register, log in, attend online examinations, submit answers, and view examination results.

The organisation has an SQA Plan that specifies requirements review, code review, software testing, defect management, change control, and maintenance of quality records.

Before the final release of the system, a software quality audit is conducted to verify whether these quality practices are being followed.

The audit is carried out through the following steps.

Step 1: Define Audit Objectives

The auditor first identifies the purpose of the audit. In this case, the objectives are to verify compliance with the SQA Plan, check whether quality activities are properly performed, and identify any non-conformities or areas requiring improvement.

Step 2: Define Audit Scope and Criteria

The audit scope includes requirements, design, coding, testing, defect management, change management, and quality documentation.

The auditor uses the SQA Plan, coding standards, testing procedures, and organisational quality procedures as audit criteria for evaluating the project activities.

Step 3: Collect Audit Evidence

The auditor collects evidence to determine whether the specified procedures have been followed. Evidence is obtained by reviewing requirements documents, design documents, code review records, test reports, defect records, and change records.

The auditor may also interview developers and testers to understand how the defined processes are actually being followed.

SGOU

Step 4: Evaluate Evidence and Record Findings

The collected evidence is compared with the audit criteria to determine compliance.

For example:

- **Requirements:** Most requirements are reviewed, but some approval records are missing — Non-conformity.
- **Code Review:** Code review records are properly maintained — Conformity.
- **Testing:** Testing is completed, but some test results are not properly documented — Non-conformity.
- **Defect Management:** Defects are properly recorded and tracked — Conformity.

The findings are documented in the audit records for further action.

Step 5: Recommend Corrective Actions

The organisation takes suitable corrective actions to address the identified non-conformities.

For example, missing requirement approval records are completed, and a standard format is introduced for recording test results. The project team is also instructed to follow these procedures consistently.

Step 6: Conduct Follow-up

After the corrective actions are implemented, the auditor verifies whether the identified problems have been properly addressed.

The auditor checks the updated requirement approvals and testing records. If sufficient evidence shows that the corrective actions are effective, the findings are closed.

Step 7: Prepare the Audit Report

The audit results are documented in an audit report. It includes the audit objectives, scope, criteria, evidence examined, findings, non-conformities, corrective actions, and follow-up results.

The report provides a formal record of the audit and helps management understand the quality status of the project.

Step 8: Audit Outcome

The audit helps the organisation identify weaknesses in requirements approval and testing documentation. Corrective actions improve these areas and strengthen compliance with the SQA Plan.

Thus, the audit supports software quality improvement, process compliance, proper documentation, and continuous improvement.

SGOU

Recap

- Software Quality Auditing is a systematic examination of software processes and quality records.
- Software quality audits verify whether software development follows defined standards and procedures.
- The main objectives of software quality auditing are to verify compliance, identify non-conformities, and improve software quality.
- Software Quality Assurance (SQA) consists of planned and systematic activities that ensure software quality.
- SQA focuses on preventing quality problems and improving software processes throughout the project.
- SQA helps reduce software defects, improve reliability, and increase customer satisfaction.
- Major SQA activities include quality planning, reviews, audits, testing, defect management, and change control.
- SQA also includes maintaining quality records, managing risks, and continuously improving software processes.
- An SQA Plan defines the quality standards, procedures, responsibilities, and activities for a software project.
- An SQA Plan helps the project team perform quality activities in a systematic and consistent manner.
- Audit activities include defining objectives, scope, and criteria before conducting the audit.
- Auditors review documents, collect evidence, and evaluate findings during the audit process.
- Audit reports record the findings and recommend suitable corrective actions.
- Follow-up activities verify that the recommended corrective actions have been successfully implemented.

SGOU

Objective Type Questions

1. What does SQA stand for?
2. What is the systematic examination of software processes called?
3. Which document specifies quality activities in a software project?
4. Which activity verifies compliance with standards and procedures?
5. Which SQA activity identifies errors before development progresses?
6. Which process helps identify software defects?
7. What are failures to meet specified requirements called?
8. Which document records the audit findings?
9. Which process supports continuous software quality improvement?
10. What is reported when a requirement is satisfied?

Answers to Objective Type Questions

1. Software Quality Assurance.
2. Audit.
3. SQA Plan.
4. Auditing.
5. Review.
6. Testing.
7. Non-conformities.
8. Audit Report.
9. SQA.
10. Conformity.

SGOU

Assignments

1. Explain the concept and objectives of Software Quality Auditing with a suitable example.
2. Describe the need for Software Quality Assurance (SQA) and explain its major activities.
3. What is an SQA Plan? Explain its importance in software development.
4. Discuss the major audit activities included in an SQA Plan.
5. Explain the Software Quality Audit case study of the Online Examination System and describe the steps followed during the audit process.

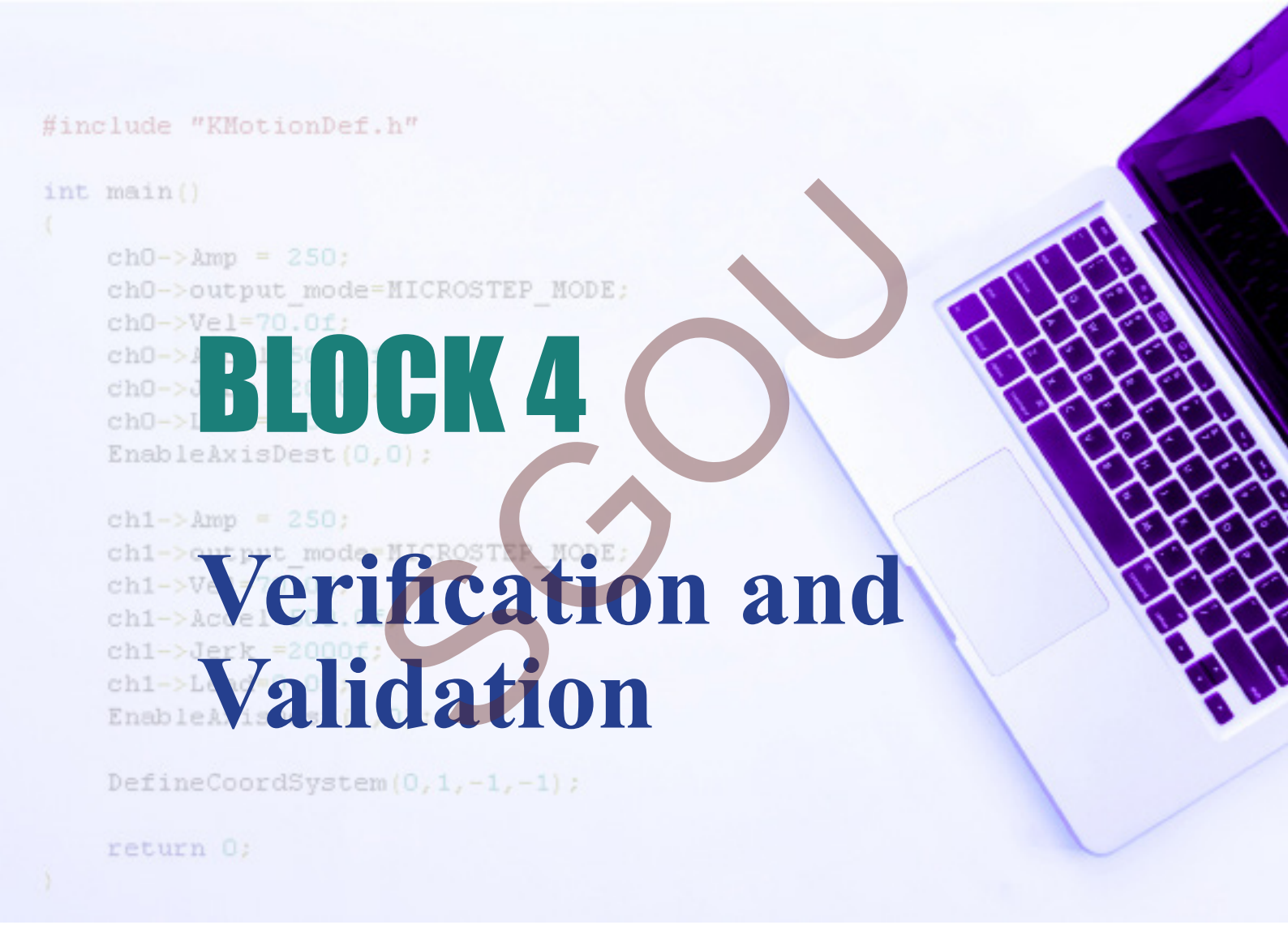
Suggested Reading

1. Hoyle, David. *ISO 9000 Quality Systems Handbook*. 7th ed., Routledge.
2. O'Regan, Gerard. *Introduction to Software Quality*. Springer, 2014.
3. International Organization for Standardization. *ISO/IEC 29110-1-1:2024: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 1-1: Overview*. ISO, 2024.
4. International Organization for Standardization. *ISO/IEC 29110-5-1-1:2025: Systems and Software Engineering—Lifecycle Profiles for Very Small Entities (VSEs)—Part 5-1-1: Software Engineering Guidelines for the Generic Entry Profile*. ISO, 2025.

SGOU

Reference

1. Galin, Daniel. *Software Quality: Concepts and Practice*. 2nd ed., Wiley-IEEE Computer Society Press, 2018.
2. O'Regan, Gerard. *Introduction to Software Process Improvement*. Springer, 2011.
3. International Organization for Standardization. *ISO 9001:2015: Quality Management Systems—Requirements*. ISO, 2015.



BLOCK 4

Verification and Validation

SGOU



V&V Standards and Models

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the meaning and importance of Verification and Validation (V&V)
- identify the main techniques used for Verification and Validation
- describe the Waterfall, V-Model, and Agile SDLC models
- explain how V&V helps to improve software quality and reduce errors

Prerequisite

Studying verification and validation standards and models is essential for understanding how software quality can be systematically planned, assessed, and assured throughout the development life cycle. These approaches help learners understand how to check whether software is developed according to specified requirements and whether the completed system satisfies user needs. They provide a structured foundation for activities such as requirements reviews, design evaluation, inspections, testing, and acceptance assessment. For example, consider a hospital patient-monitoring system, where an incorrect reading or delayed alert could seriously affect patient care.

Verification activities can help ensure that the requirements, design, and implementation correctly address the specified functions, while validation activities can determine whether the completed system performs as expected in its intended environment.

Understanding these approaches enables learners to recognise the importance of systematic quality assessment rather than relying only on informal checking or final-stage testing. Therefore, this knowledge helps reduce defects, improve software reliability, and increase confidence that a software system is safe, effective, and suitable for its intended purpose

SGOU

Keywords

Requirements Traceability, Acceptance Criteria, Static Analysis, Integration Testing, Regression Testing, Stakeholder Feedback, Defect Detection

Discussion

Role of Verification and Validation (V&V) in the Software Development Life Cycle

Verification and Validation (V&V) are important activities within the software life cycle that help provide confidence that the software product and its related work products satisfy their intended requirements and needs. Although verification and validation are closely related, they address different questions. Verification is concerned with determining whether a work product or software system has been developed correctly with respect to the specified requirements and applicable criteria. It can involve examining requirements, design descriptions, code, test-related work products, and other development outputs to determine whether they conform to the specified conditions. Validation, on the other hand, is concerned with determining whether the resulting software is suitable for its intended use and satisfies the needs of the relevant stakeholders.

For example, consider the development of an online university admission system. During verification, the development team may examine the requirements and design to determine whether the specified admission rules, eligibility conditions, fee calculations, and application-processing functions have been correctly represented in the system. The team may also check whether the implemented functions conform to the approved specifications. During validation, the system is evaluated in its intended context to determine whether prospective learners and university staff can actually use the system effectively to complete activities such as registration, document submission, fee payment, and application tracking. Thus, verification provides evidence that the system has been developed according to the specified requirements, while validation provides evidence that the resulting system is appropriate for its intended purpose.

V&V also contributes to early identification of defects and reduction of development risks. If an incorrect requirement, missing condition, or inconsistency is identified during the review of requirements or design, it can generally be corrected before it propagates into implementation and testing. For example, if an admission system requirement incorrectly states that a particular category of learner is ineligible for a programme, identifying this problem during requirements verification is considerably easier than discovering it after the software has been implemented and used by applicants. Similarly, validation may reveal that a technically correct feature does not adequately support the actual needs of users. Therefore, applying verification and validation throughout the software life cycle helps improve the quality and reliability of the resulting software and reduces the possibility of costly corrections later in development.

SGOU

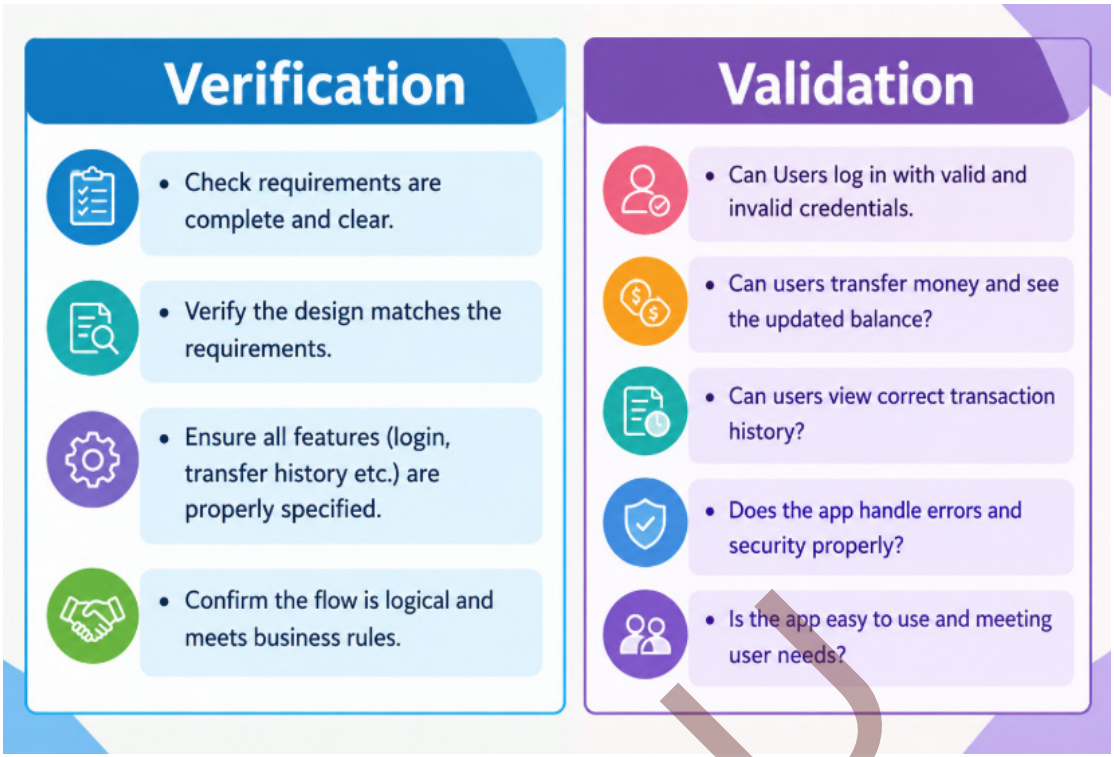


Fig. 4.1.1 Comparison of Verification and Validation Using an E-Wallet Application

4.1.1 Verification

Verification is the process of determining whether a software work product or system element has been developed correctly in accordance with its specified requirements and design information. In simple terms, verification asks, **"Are we building the product right?"** It focuses on checking whether the outputs produced during software development conform to the requirements, specifications, standards, and other applicable conditions established for the system. Verification can be carried out throughout the development process rather than being limited to the final stage. For example, when developing a banking application, the requirements may specify that a customer should be able to transfer money between accounts only after successful authentication. During verification, the requirements and design documents can be examined to determine whether the proposed design adequately addresses authentication, account selection, transaction authorisation, and transaction recording. Similarly, a review of a database design can determine whether all the data items required by the specified requirements have been included. Such activities help identify omissions, inconsistencies, or other defects in work products before they propagate to later stages of development.

Verification may involve appropriate examination, review, analysis, inspection, or other evaluation activities depending on the work product and the conditions that need to be demonstrated. For example, a design document may be reviewed to check its consistency with the approved requirements, while source code may be examined to determine whether it conforms to applicable coding standards and design decisions. By identifying problems in intermediate work products, verification provides an opportunity to correct them before they affect subsequent development activities. Thus,

SGOU

verification contributes to producing software work products that conform to their specified requirements and intended technical characteristics. See fig. 4.1.1.

4.1.2 Validation

Validation is the process of evaluating a software product to determine whether it satisfies its intended use and the needs of the users and other relevant stakeholders. It focuses on the question, **"Are we building the right product?"** In validation, the developed software is evaluated in conditions that represent its intended use so that it can be determined whether the product fulfils its intended purpose.

Validation activities may involve executing the software and examining its behaviour, outputs, and suitability for the intended users. For example, consider an online shopping application developed with features such as user login, product search, shopping cart, and online payment. During validation, the application is used in realistic scenarios to determine whether users can successfully log in, find the required products, add them to the cart, complete payment, and receive the expected order confirmation. If users find that the product-search facility does not support the way they normally search for products, the validation activity may reveal that the software does not adequately satisfy the intended user needs, even if the feature technically operates according to its specified design. Thus, validation provides confidence that the developed software is suitable for its intended use and fulfils the needs for which it was developed. See fig.4.1.1.

V&V activities are applied throughout the SDLC to ensure quality at every stage. See fig. 4.1.2.

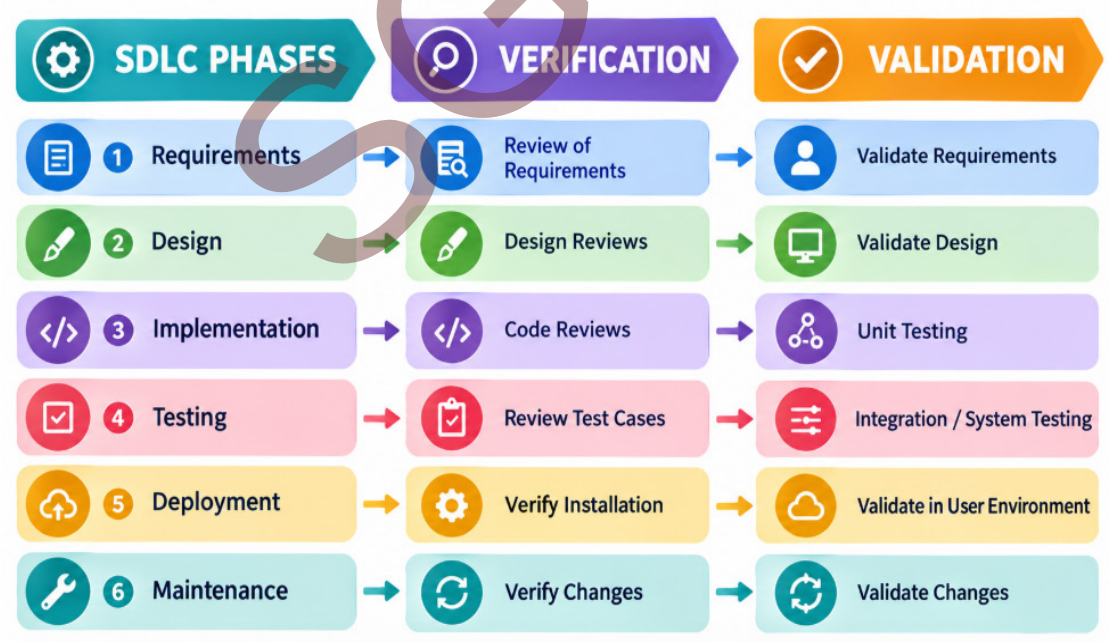


Fig. 4.1.2 SDLV mapping with V&V

SGOU

4.1.3 Verification and Validation Across the Software Life Cycle

During the requirements phase, verification focuses on checking whether the documented requirements are clear, complete, consistent, and sufficiently detailed for further development. For example, if a requirement states that "the system should respond quickly," verification would identify that the requirement is ambiguous because the term "quickly" is not measurable. The requirement could instead specify a measurable response time. Validation, on the other hand, considers whether the documented requirements actually represent the needs and expectations of the stakeholders. For example, if users require a student registration system to allow registration through both computers and mobile devices, validation would confirm that this need has been properly captured in the requirements.

During the design phase, verification examines whether the proposed design correctly implements the approved requirements and whether the design elements are consistent with one another. For example, if a requirement specifies that only authorised users can access student records, verification of the design would check whether appropriate authentication and authorisation mechanisms have been incorporated. Validation at this stage considers whether the proposed design is suitable for the intended system and operational environment. For example, a design may technically satisfy the requirements but still be unsuitable if it is too complex for the intended users or cannot operate effectively within the available infrastructure.

During the implementation phase, verification focuses on determining whether the implemented software correctly conforms to the approved design and applicable requirements. Activities such as code reviews, inspections, and walkthroughs can be used to identify defects or deviations. For example, if the design specifies that a password must contain at least eight characters, a code review can verify whether the implemented password-validation logic correctly enforces this rule. Validation progressively examines whether the implemented software provides the intended functionality. For example, executing a unit-level function that validates passwords can help determine whether the implemented behaviour produces the expected result for valid and invalid inputs.

During the testing phase, verification focuses on checking whether the testing activities and test work products adequately address the specified requirements and whether the planned tests are properly designed and executed. For example, if a requirement specifies that a system must support 1,000 simultaneous users, verification can examine whether an appropriate performance test has been designed to evaluate this requirement. Validation focuses on determining whether the integrated system behaves as intended in its intended context of use. For example, system-level testing can determine whether users can successfully complete the complete registration process, from entering personal details to receiving confirmation, under realistic operating conditions.

During deployment and maintenance, verification is concerned with ensuring that modifications, configurations, updates, and related work products conform to the approved requirements and applicable specifications. For example, before deploying

SGOU

a software update, verification can confirm that the correct version has been built, the approved configuration has been used, and the intended changes have been implemented. Validation considers whether the modified or deployed system continues to satisfy its intended purpose in the operational environment. For example, after introducing a new online payment facility, validation can determine whether users can successfully complete payments in the actual operational environment and whether the functionality meets the intended business and user needs.

4.1.4 Techniques Used in Verification

Verification is concerned with examining software work products to determine whether they have been developed correctly and whether they conform to their specified requirements. It can be performed through activities such as reviews, inspections, and walkthroughs, which help identify defects before they progress to later stages of development. For example, when a requirement specification is prepared for an online examination system, a review may be conducted to check whether every stated requirement is clear, complete, consistent, and properly documented.

Similarly, a formal inspection involves a structured examination of a document or other work product by a group of qualified reviewers. The reviewers systematically examine the material to identify errors, omissions, inconsistencies, or ambiguities. For example, during an inspection of a software design document, the team may discover that a requirement for password recovery has been specified, but no corresponding design component has been included.

A walkthrough is another verification technique in which the person who prepared the work product explains it step by step to other team members. The participants examine the work from different perspectives and raise questions or identify possible problems. For example, a developer may conduct a walkthrough of the design of a student registration system, explaining how student details are entered, validated, stored, and retrieved. The reviewers may then identify a missing validation rule or an inconsistent data flow. These techniques help detect problems early, when correcting them is generally easier and less costly than discovering them after implementation.

4.1.5 Techniques Used in Validation

Validation involves evaluating the software to determine whether it satisfies its intended purpose and the needs of its users. It is generally carried out by executing the software under defined conditions and observing its behaviour and results.

Unit testing focuses on individual software components or modules to determine whether each component performs its intended function correctly. For example, in an online shopping application, the function that calculates the total price of items in a shopping cart can be tested separately to ensure that it produces the correct amount.

Integration testing evaluates whether two or more combined modules interact correctly. For example, after testing the shopping-cart module and payment module separately, integration testing can verify whether the correct order amount is successfully transferred from the shopping cart to the payment system.

SGOU

System testing evaluates the complete software system as an integrated product to determine whether its overall functionality operates as intended. For example, an online examination system may be tested from student login through question answering, submission, result processing, and report generation.

Acceptance testing is performed to determine whether the completed software satisfies the needs and expectations of its intended users or customers. For example, before releasing an online admission system, university staff may use the system to verify that applicants can register, upload required documents, make payments, and complete the application process according to the intended requirements.

4.1.6 Limitations of Verification and Validation (V&V)

Although Verification and Validation (V&V) are important for improving software quality and reducing the risk of defects, they also have certain limitations. V&V activities require additional time, effort, and resources, which can increase the overall cost of software development. For example, a banking application may require repeated reviews, test execution, security checks, and validation of different transaction scenarios before release. These activities provide greater confidence in the software but also require additional personnel, testing infrastructure, and development time.

V&V can also be time-consuming, particularly for large and complex software systems. Different components, interfaces, requirements, and system behaviours may need to be examined separately and then evaluated together. For example, in an online examination system, the login facility, question delivery, examination timer, answer submission, result generation, and database operations may all require separate verification and validation activities. Consequently, thorough V&V may extend the development schedule.

Another limitation is that V&V cannot guarantee that every defect will be identified. Testing and review activities can reveal many defects, but some problems may remain undetected because they occur only under unusual conditions or combinations of events that were not included in the verification or validation activities. For example, an application may function correctly during normal testing but experience a failure when thousands of users access it simultaneously. Therefore, successful V&V should not be interpreted as proof that the software is completely defect-free.

The effectiveness of V&V also depends partly on the knowledge, experience, and judgement of the people performing the activities. An experienced reviewer may identify an ambiguous requirement that another reviewer overlooks. Similarly, an experienced tester may design test scenarios that expose problems under unusual conditions. This means that human judgement can influence the effectiveness of V&V activities.

V&V can become more challenging when a system is highly complex, or its requirements change frequently. When requirements are modified, previously verified designs, implementations, and test results may need to be reviewed again. For example, if an e-commerce system introduces a new payment method after testing has already been

SGOU

completed, the related requirements, interfaces, security considerations, and test cases may need to be re-examined. Frequent changes can therefore increase the V&V effort.

There is also a practical need to maintain a balance between thoroughness and development time. Excessive or unnecessarily repeated verification and validation activities can delay software delivery, particularly when the associated risk is low. For example, repeatedly performing the same extensive tests on an unchanged and well-understood component may consume resources without providing proportional benefits. Therefore, V&V activities should be appropriately planned and applied according to the characteristics, risks, and objectives of the software system.

V&V increases confidence in the correctness, quality, and suitability of software, but it cannot provide an absolute guarantee of defect-free software or complete correctness. Its purpose is to identify and reduce risks and defects through appropriate verification and validation activities. The limitations of V&V therefore highlight the importance of selecting suitable activities, allocating adequate resources, maintaining appropriate documentation and traceability, and performing V&V throughout the relevant stages of the software life cycle.

4.1.7 Types of SDLC Models

SDLC models and methodologies can be classified into different categories based on various criteria and approaches to software development.

There are two fundamental categories of software development life cycle models:

- Heavyweight or Predictive Models
- Lightweight or Agile Models

4.1.7.1 Heavyweight (Predictive) Models

Heavyweight processes are based on a predefined scope of work, where the major requirements, activities, and deliverables are determined in advance. For this reason, they are also referred to as predictive models. These approaches generally require substantial involvement from development teams and place considerable emphasis on detailed documentation and formal processes.

Traditionally, organisations preferred heavyweight approaches for managing and structuring large and extensive software projects, particularly when requirements were clearly established at the beginning. However, their popularity has decreased in situations where requirements change frequently, as the additional procedures and documentation can reduce flexibility and project efficiency.

Predictive (heavyweight) models include:

- Waterfall Model
- V-shaped Model

There are many other Predictive models as well.

SGOU

4.1.7.2 Lightweight (Agile) Models

Lightweight processes, commonly known as Agile or Adaptive Approaches, provide greater flexibility in comparison with heavyweight models. They offer a balanced approach between excessive process control and the complete absence of structured development practices. Agile methodologies generally require less documentation and emphasise iterative development, continuous feedback, collaboration, and the ability to respond to changing requirements.

For example, an adaptive approach can accommodate changing customer requirements, such as the need to modify or enhance an application continuously throughout its development. In contrast, a predictive SDLC model may become increasingly difficult to manage when unexpected changes are introduced after the development plan has been established.

Heavyweight methodologies are generally more appropriate when requirements are clearly defined, project activities can be planned in advance, and large development teams are involved. Agile approaches are more suitable when requirements are expected to change frequently, and development is carried out by relatively small, collaborative teams.

Adaptive SDLC Models

Adaptive SDLC models, commonly associated with Agile development, are increasingly used for software projects where requirements may change during development. Unlike rigid approaches that depend heavily on following a predefined plan, adaptive models emphasise flexibility, continuous communication, collaboration, and the ability to respond quickly to changing needs. They allow development teams to adjust their plans and priorities as new requirements, feedback, or challenges emerge during the project.

Several Agile approaches can be implemented as adaptive SDLC models. The commonly used approaches include Scrum, Extreme Programming (XP), and Kanban. Each approach provides different practices for managing development activities while maintaining flexibility and continuous improvement.

4.1.8 Details of SDLC Models

Software Development Life Cycle (SDLC) models provide structured approaches for planning, developing, testing, deploying, and maintaining software systems. Different SDLC models, such as Waterfall, Iterative, Incremental, Spiral, V Model, and Agile, organise these activities in different ways according to project requirements and constraints. Each model defines how requirements, design, implementation, verification, validation, and maintenance activities are performed throughout the software life cycle. For example, the Waterfall model follows a sequential approach, while the Agile model supports frequent development cycles and continuous feedback. Understanding different SDLC models helps learners select an appropriate development approach and understand how verification and validation activities can be integrated into the software development process.

SGOU

4.1.8.1 Waterfall SDLC Model

The Waterfall is a cascade SDLC model that presents the development process as a flow, moving step by step through the phases of analysis, projecting, realisation, testing, implementation, and support. This SDLC model includes the gradual execution of every stage. Waterfall implies strict documentation. The features expected of each phase of this SDLC model are predefined in advance.

This approach allows avoiding many mistakes that may appear because of insufficient control over the project. It results in pervasive documentation development. It is beneficial to the developers who may be working with the product in the future, but it takes a long time to write everything down.

In some cases, the feedback loop is included. It allows making short reviews of each stage's result and applying some minor amendments. This loop enables specialists to return to the previous phase for a short period. If something significant changes in the initial plan, a team should wait until the very last stage to return to the beginning and pass all software life cycle phases again.

4.1.8.2 Advantages of the Waterfall Model

The Waterfall Model is relatively simple to understand and use because the development process is divided into clearly defined phases. Each phase has a specific purpose, activities, and expected outcomes. For example, a project may proceed through requirements analysis, system design, implementation, testing, deployment, and maintenance in a clearly defined sequence. This makes it easier for developers, project managers, and other stakeholders to understand what needs to be completed at each stage.

The model also provides management simplicity because of its structured and sequential nature. Each phase has defined deliverables and can be reviewed before the project moves to the next phase. For example, after completing the requirements phase, the requirements document can be reviewed and approved before design activities begin. This provides project managers with clear checkpoints for monitoring progress and identifying whether a phase has been completed as planned.

Another important advantage is that the development stages proceed sequentially. The completion of one phase provides the basis for starting the next phase. For example, once the system requirements have been finalised, developers can use those requirements to prepare the system design. Similarly, the completed design serves as the basis for implementation. This structured progression can make the overall development process easier to organise and monitor.

The Waterfall Model is particularly suitable for small or medium-sized projects where requirements are well understood and unlikely to change significantly. For example, if an organisation needs a simple payroll system with clearly established rules, salary calculations, employee records, and reporting requirements, the requirements can be defined in detail before development begins. In such situations, the sequential nature of the Waterfall Model can be advantageous.

SGOU

The model also makes it easy to identify important milestones or key points in the development life cycle. Each phase can be treated as a milestone, such as completion and approval of requirements, completion of system design, completion of coding, completion of testing, and system deployment. These milestones help project managers evaluate progress and compare the actual status of the project with the planned schedule.

The structured nature of the Waterfall Model makes it easier to classify and prioritise development activities. Tasks can be organised according to the phase to which they belong and their dependencies on other activities. For example, requirements-related tasks can be completed before design tasks, while testing activities can be planned based on the completed implementation. This clear organisation helps teams understand task dependencies and allocate resources more systematically.

4.1.8.3 Disadvantages

The Waterfall model has several limitations because development proceeds through a sequence of predefined stages, with each stage generally completed before the next one begins. One major disadvantage is that the complete software product becomes available only after the final stage has been completed. Users may therefore have to wait until the end of the development cycle before they can use or evaluate the complete system. For example, if an organisation develops a university management system using the Waterfall model, students and staff may not be able to use the actual system until the requirements, design, implementation, testing, and deployment stages have all been completed.

Another limitation is the high level of risk and uncertainty, particularly when the requirements are not fully understood at the beginning of the project. If an important requirement changes after the design or implementation stage has been completed, modifying the earlier work may require considerable time and cost. For example, if a university initially specifies that an admission system should support only online applications but later decides that it must also support offline applications and document verification, significant changes may be required in the design and implementation.

The Waterfall model may also be less suitable for complex projects and projects involving frequently changing requirements. Complex software systems often involve interactions among many components, and understanding all requirements accurately at the beginning can be difficult. Object-oriented and continuously evolving projects may require frequent refinement of requirements, design, and implementation, whereas the traditional Waterfall approach emphasises completing one stage before proceeding to the next.

It can also be inappropriate for long-term projects in environments where requirements, technologies, or business conditions are likely to change. In a project that takes several years to complete, the requirements identified at the beginning may no longer completely reflect the users' needs by the time the software is ready for deployment. For example, a long-term banking application developed using requirements defined several years earlier may need substantial changes because of new payment technologies, security requirements, or regulatory changes.

SGOU

Another difficulty is that progress can be difficult to assess accurately while development is still in progress. Completing a phase does not necessarily mean that the corresponding work is completely correct or that later phases will proceed without problems. A project may appear to be progressing according to schedule during the requirements and design stages, but serious issues may become visible only when implementation and testing begin. Therefore, measuring meaningful functional progress can be more difficult than simply measuring the completion of individual development phases.

A further limitation is that integration is generally performed relatively late in the development process. When individual components are developed separately and brought together only after substantial development has been completed, interface problems and compatibility issues may remain undetected for a long time. For example, in an online examination system, the authentication module, question-management module, examination module, and result-processing module may work correctly when considered individually but encounter problems when integrated. If such problems are discovered only near the end of the project, correcting them can require significant rework and may affect the project schedule.

Thus, although the Waterfall model provides a structured and systematic development approach, its sequential nature can make it difficult to respond efficiently to changing requirements, discover integration problems early, and obtain usable software during the earlier stages of a project.

4.1.8.4 V-Model (Verification and Validation Model)

The V-Model is a Software Development Life Cycle (SDLC) approach that emphasises verification and validation throughout the software development process. It is structured in a V-shaped pattern, in which each development phase is systematically linked to a corresponding testing phase. This approach helps identify defects at an early stage and improves the overall quality and reliability of the software. See fig. 4.1.3.

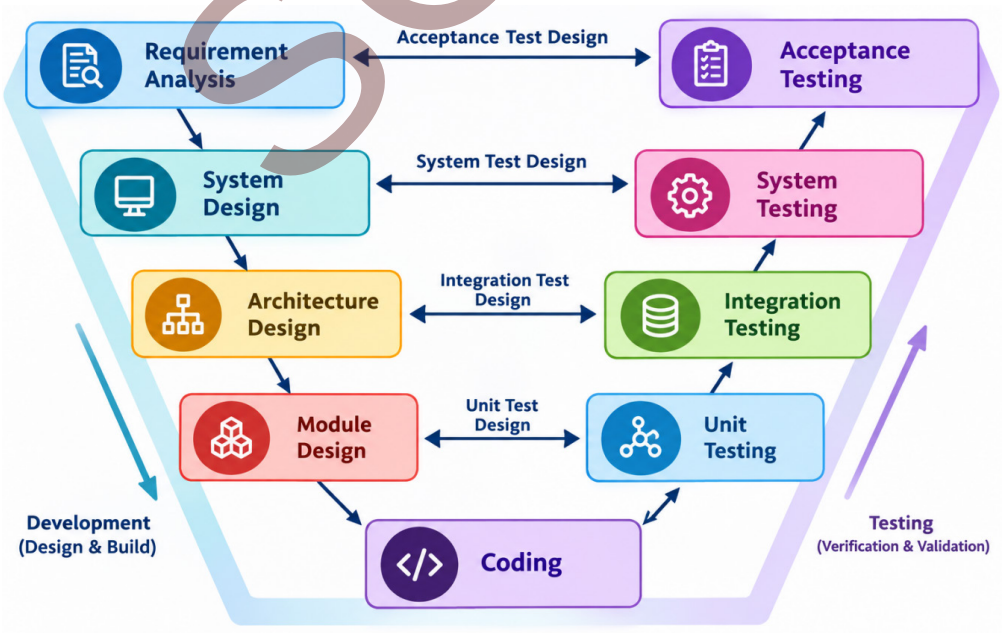


Fig. 4.1.3 The V-Model

4.1.8.5 Phases of the SDLC V-Model

The V-Model is a structured software development approach in which development activities are systematically linked with corresponding testing activities. It emphasises verification during the development stages and validation through testing after implementation. The model follows a sequential and systematic approach, with each development phase having a corresponding testing phase.

I. Verification Phases

Verification focuses on examining whether the software is being developed according to the specified requirements and design. It primarily involves activities that can be performed without executing the software, such as requirement reviews, design reviews, inspections, and static analysis.

The main objective is to ensure that each development output is correct, consistent, and aligned with the requirements before moving to the next stage.

- **Business Requirement Analysis**

The process begins by identifying and understanding the customer's business needs and expectations. These requirements are then analysed to establish the scope and objectives of the software project. At this stage, acceptance test plans can also be prepared based on the identified business requirements. For example, in an online banking system, the requirements may include fund transfer, balance enquiry, online bill payment, and secure customer authentication. These identified requirements provide a clear basis for planning acceptance tests and subsequently evaluating whether the completed system satisfies the intended business needs and user expectations.

II. System Design

During system design, the overall structure and major components of the software system are defined by considering both high-level architecture and detailed design aspects. At this stage, the relationships and interactions among the major components are also established, and system-level testing requirements are identified to ensure that the proposed design can satisfy the specified system requirements. For example, in an e-commerce application, the system design may define major components such as user management, product management, shopping cart, payment processing, and order management, while also considering how these components interact to support the overall functioning of the system.

III. Architectural Design

Architectural design defines the overall structure of a system and specifies how its major components or modules interact with one another. These interactions provide a basis for planning integration testing requirements, particularly for verifying whether the components communicate and work together as intended. For example, in a hospital management system, separate modules may be designed for patient registration, appointments, billing, laboratory services, and medical records. The architectural design specifies how these modules communicate with one another, allowing integration testing

SGOU

to focus on the correctness of data exchange and interactions between the connected modules.

IV. Module Design or Low-Level Design (LLD)

At this stage, the internal structure and detailed behaviour of individual modules are designed. The inputs, outputs, processing logic, data structures, and interfaces of each module are defined, and unit test cases can be prepared based on the detailed design. For example, in a banking application, the fund-transfer module may be designed to verify the account number, check the available balance, calculate the transaction amount, update account records, and generate a transaction confirmation. These detailed specifications provide a basis for verifying whether each module is implemented according to its intended design and for developing appropriate unit tests to check its behaviour.

V. Coding

Coding is the implementation stage in which developers convert the approved design into executable software. During this stage, the design specifications are translated into program instructions using appropriate programming languages, coding standards, and development practices. For example, if the design of a login module specifies that users must provide a valid username and password, the developers implement these requirements through suitable program logic. Code reviews and static analysis can then be performed to identify defects, inconsistencies, or violations of coding standards before the software is executed.

4.1.8.6 Validation Phase

Validation focuses on evaluating the implemented software by executing it and determining whether it behaves as expected and satisfies the intended requirements. It primarily involves dynamic testing, including functional and non-functional testing.

The validation activities progress from testing individual software units to testing integrated modules, the complete system, and finally the system from the user's perspective.

I. Unit Testing

Unit testing examines individual program units or modules separately. The objective is to determine whether each unit performs its intended function and to identify defects at the code or module level. For example, in an online shopping application, the function that calculates the total price of items in a shopping cart can be tested independently to verify that the calculation produces the correct result for different quantities and prices.

II. Integration Testing

After individual modules have been tested, the related modules are integrated and tested together to ensure that they work correctly as a combined system. Integration testing focuses particularly on the interfaces, data exchange, and interactions between modules. For example, in an online shopping system, the shopping-cart module communicates with the payment module during a transaction. Integration testing verifies whether the correct order amount is transferred from the shopping cart to the payment system and

SGOU

whether the payment result is correctly returned to the shopping-cart system. Thus, integration testing helps identify problems that may not be detected when the modules are tested independently.

III. System Testing

System testing evaluates the complete integrated application as a whole. It examines whether the different components and subsystems work together correctly and whether the overall system satisfies its specified functional and non-functional requirements. For example, in an online examination system, system testing may evaluate the complete flow from student login and examination registration to question delivery, answer submission, automatic evaluation, result generation, and report generation. Instead of testing these functions independently, system testing considers how they operate together as an integrated system and determines whether the complete application behaves as expected under the specified conditions.

IV. User Acceptance Testing (UAT)

User Acceptance Testing (UAT) is conducted to determine whether the completed system is acceptable to its intended users and suitable for use in an environment that closely resembles the actual operational environment. During UAT, users perform realistic tasks using the system to determine whether it supports their actual requirements and operational procedures. For example, before deploying a university admission system, admission officers may perform actual admission-related activities using the system to verify whether applications can be processed correctly, documents can be reviewed, payments can be confirmed, and admission reports can be generated according to the university's established procedures. This helps determine whether the system is ready for acceptance and practical use by its intended users.

4.1.8.7 Challenges of Using V-Model

Modern software systems are increasingly complex and are often developed under strict time and resource constraints. These conditions make it important to establish clear requirements and consider testing activities from the early stages of development.

A major challenge is accurately identifying and refining user and business requirements. Ambiguous or incomplete requirements can result in misunderstandings between customers, developers, and testers.

Another challenge is ensuring that the software design and implementation remain consistent with the approved requirements. Changes introduced during development must also be carefully controlled so that they do not introduce inconsistencies between requirements, design, implementation, and testing.

The final challenge is determining whether the developed application actually satisfies the intended business purpose. A system may technically function correctly but still fail to meet users' practical expectations. Therefore, both verification and validation are important for establishing confidence in the resulting software.

SGOU

4.1.8.8 Importance of the V-Model

The V-Model provides a structured relationship between software development and testing activities. Instead of postponing testing until software development is completely finished, testing requirements are considered alongside the corresponding development activities.

- **Early Defect Detection:** The model encourages defects to be identified as early as possible through requirement reviews, design reviews, code reviews, and other verification activities. Detecting a requirement or design defect early generally makes it easier to correct than discovering the same problem after implementation.
- **Correspondence Between Development and Testing:** Each major development activity has a corresponding testing activity. Business requirements are associated with acceptance testing, system design with system testing, architectural design with integration testing, and module design with unit testing.
- **Avoidance of End-Stage Testing:** The V-Model discourages the idea of performing all testing only after coding has been completed. Testing considerations are introduced during the development process, allowing test planning to begin early.
- **Improved Coordination:** Because development and testing activities are linked, developers, testers, analysts, and other stakeholders can establish a clearer understanding of what needs to be produced and how the resulting work will be evaluated.
- **Improved Software Quality:** The systematic combination of verification and validation helps identify defects, confirm compliance with requirements, and evaluate whether the completed system is suitable for its intended purpose.

4.1.8.9 Principles of the V-Model

- **Large to Small:** The V-Model generally moves from broader requirements and system-level considerations toward increasingly detailed design and implementation. The development process becomes progressively more detailed as it moves toward coding. Testing then proceeds from individual components toward the complete system.
- **Data and Process Integrity:** The design should maintain consistency between data and the processes that operate on that data. Requirements concerning data and processing should be considered throughout the relevant development stages.
- **Scalability:** The V-Model can be adapted to software projects of different sizes and levels of complexity. The degree of formality and detail applied to the activities can vary according to the characteristics and requirements of the project.
- **Cross-Referencing:** An important characteristic of the V-Model is the relationship between development requirements and corresponding testing activities. Requirements can be linked to appropriate test activities so that the project team can determine how each requirement will eventually be evaluated.

SGOU

4.1.8.10 Applications of the V-Model

The V-Model is particularly useful in domains where requirements need to be clearly established, and software quality, reliability, safety, or compliance is important.

- **Healthcare Systems:** The model can be applied to hospital management systems, medical imaging applications, patient monitoring systems, and other healthcare software. Thorough verification and validation are particularly important because software failures may have serious consequences.
- **Banking and Financial Systems:** Banking applications, ATM software, payment systems, and internet banking platforms require accurate processing, security, and reliability. The V-Model provides a structured approach for relating requirements to testing activities.
- **Aerospace and Aviation:** Software used in aircraft navigation, flight control, and monitoring systems requires extensive engineering and testing because failures can have significant consequences. A structured development and testing approach can therefore be valuable.
- **Automotive Systems:** Automotive software, including systems associated with anti-lock braking, airbags, driver assistance, and engine control, requires careful development and testing. The V-Model can provide a systematic framework for connecting requirements and design activities with corresponding verification and validation activities.

4.1.8.11 Advantages of the V-Model

- Adopts a systematic and well-organised phase-by-phase development approach.
- Works well for small projects where requirements are clearly defined and unlikely to change.
- Gives significant importance to verification and validation from the early stages of development.
- Provides a structured workflow with a strong emphasis on testing and quality assurance.

4.1.8.12 Disadvantages of the V-Model

- Offers limited flexibility when requirements change during development.
- May require considerable time because of extensive documentation, reviews, and testing activities.
- May not be appropriate for highly complex or high-risk software projects.
- Does not naturally support iterative development or frequent changes to the system.

SGOU

4.1.8.13 Agile Software Development Methodology

Agile Software Development Methodology is an approach to software development that emphasises iterative and incremental development. Similar to other software development approaches such as the Waterfall, V-Model, and Iterative models, Agile provides a structured way to develop software. Agile focuses strongly on delivering working software in small increments and responding quickly to changing requirements. The term Agile refers to the ability to move or respond quickly, which reflects the methodology's emphasis on flexibility and rapid adaptation to change.

Agile methodology divides a software project into small, manageable increments or iterations rather than developing the entire system as a single process. Each iteration involves activities such as planning, development, testing, and review, resulting in a usable software increment. Feedback from customers and other stakeholders is incorporated into subsequent iterations, allowing the product to evolve according to changing requirements.

Agile gives importance to flexibility, continuous collaboration, customer involvement, and customer satisfaction. Instead of relying entirely on requirements defined at the beginning of a project, Agile allows requirements to be refined as the project progresses. This makes it particularly useful for projects where requirements may change frequently.

4.1.8.14 Suitability of Agile Methodology

Agile is particularly suitable for projects where requirements are expected to change, and frequent stakeholder feedback is required. Its iterative nature allows teams to respond to new requirements and continuously improve the software. The same flexibility can create challenges such as scope changes, increased coordination requirements, and difficulties in maintaining extensive documentation when documentation is not given sufficient attention.

Agile is generally appropriate for dynamic projects that require adaptability, continuous feedback, and incremental delivery. Projects that depend heavily on fixed requirements, rigid schedules, strict regulatory documentation, or highly structured development procedures may require additional planning and control mechanisms alongside Agile practices.

4.1.8.15 Types of Agile Methodologies

Several frameworks and approaches are commonly associated with Agile software development. Scrum organises development into short iterations called sprints and emphasises defined roles, regular meetings, and continuous review. Kanban focuses on visualising workflow and managing the continuous movement of tasks through different stages of development. Lean emphasises eliminating unnecessary activities, reducing waste, and maximising customer value.

Other Agile approaches include Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature-Driven Development (FDD), and Crystal. For large organisations and complex projects, the Scaled Agile Framework (SAFe) provides

SGOU

an approach for applying Agile principles across multiple teams and organisational levels.

4.1.8.16 Advantages of Agile Methodology

One of the major advantages of Agile is its focus on customer value. Development work is divided into smaller tasks, allowing teams to prioritise important features and deliver useful functionality early. Customer feedback can then be incorporated into subsequent iterations, helping the development team ensure that the product continues to meet user expectations.

Agile also contributes to team motivation and ownership. Development teams are generally encouraged to collaborate, make decisions, and take responsibility for their assigned work. This collaborative environment can improve team involvement and create a stronger sense of responsibility for the quality and progress of the project.

Another important advantage is continuous stakeholder collaboration. Customers, product owners, developers, testers, and other stakeholders can communicate regularly throughout development. Continuous interaction helps clarify requirements, identify misunderstandings, and obtain feedback before the project reaches its final stage.

Agile also supports early and continuous delivery of working software. Instead of waiting until the entire project is completed, teams can deliver functional increments at regular intervals. This provides stakeholders with visible evidence of progress and creates opportunities to identify and address problems at an early stage.

Software quality is also an important aspect of Agile development. Practices such as continuous integration, automated testing, regular reviews, and frequent feedback can help identify defects earlier in the development process. By incorporating quality-related activities throughout development, the team can reduce the likelihood of discovering major problems only near the end of the project.

4.1.8.17 Disadvantages of Agile Methodology

Despite its benefits, Agile also has certain limitations. One significant limitation is reduced predictability. Since requirements and priorities may change during development, it can be difficult to determine the exact project completion date, final scope, or total cost at the beginning of the project. This can create difficulties for stakeholders who require fixed schedules and budgets.

Agile also has a strong dependency on customer and stakeholder involvement. Regular feedback is important for making appropriate decisions about future iterations. If customers or stakeholders are unavailable, provide delayed feedback, or lack sufficient understanding of the requirements, development activities may be affected.

Scaling Agile can become challenging when a project involves a large number of teams or highly complex systems. As the number of teams increases, maintaining effective communication, coordination, and consistency becomes more difficult. Additional mechanisms may therefore be required to coordinate development across teams.

SGOU

Agile is also highly dependent on effective team dynamics. Since Agile teams are expected to collaborate closely and take responsibility for many development decisions, inadequate communication, insufficient technical expertise, or lack of experience can negatively affect productivity and software quality.

Another limitation is the additional planning, coordination, and communication effort associated with Agile practices. Regular meetings, reviews, planning activities, and stakeholder interactions require time and resources. For very small projects or projects with extremely tight deadlines, this additional overhead may become a challenge.

4.1.8.18 Verification and validation in Waterfall model

In the Waterfall model, verification and validation are performed at different stages of the software development process. **Verification** mainly involves static techniques that examine whether the software work products are being developed correctly according to the specified requirements. It generally does not require executing the software. Common verification techniques include requirements review, requirements inspection, design review, architecture review, code review, static code analysis, and documentation review. For example, during a requirements review for an online banking system, the team may examine whether requirements such as secure login, fund transfer, and transaction confirmation are clearly and consistently specified. Similarly, a design review can determine whether the proposed system architecture adequately addresses the approved requirements, while code reviews and static analysis can identify programming errors and coding issues before the software is executed.

Validation, on the other hand, involves executing the developed software to determine whether it satisfies the specified requirements and is suitable for its intended use. Common validation techniques include unit testing, integration testing, system testing, acceptance testing, functional testing, performance testing, security testing, and usability testing. For example, in an online shopping application, unit testing can verify individual modules such as the shopping-cart calculation, integration testing can examine the interaction between the shopping cart and payment modules, and system testing can evaluate the complete application. Finally, acceptance testing can be performed by the intended users or customers to determine whether the system meets their practical requirements.

Thus, in the Waterfall model, verification focuses mainly on reviewing and evaluating work products before execution, while validation focuses mainly on testing the implemented software through execution. Unlike the V-Model, where development and corresponding testing activities are explicitly paired, the traditional Waterfall model follows a sequential progression in which formal software testing is generally concentrated after implementation.

4.1.8.19 Verification and Validation in V-model

In the V-Model, verification and validation activities are explicitly associated with the corresponding development and testing stages. Verification mainly examines whether each development work product is correctly produced according to the specified requirements, while validation evaluates the implemented software through execution

SGOU

to determine whether it satisfies those requirements and the intended user needs.

Verification techniques

During the verification side of the V-Model, common techniques include requirements review, requirements analysis, specification review, design review, architecture review, inspections, walkthroughs, code review, and static analysis. These activities are generally performed without executing the software. For example, during Business Requirement Analysis, the requirements are examined for correctness, completeness, consistency, and clarity, and acceptance test requirements can be identified. During System Design, the design is reviewed against the system requirements. During Architectural Design, the architecture and interactions among modules are examined, while during Module Design, the detailed design of individual modules is reviewed. During coding, code reviews and static analysis can be used to identify defects before execution.

Validation techniques

On the validation side, the implemented software is executed and tested progressively. The major techniques are unit testing, integration testing, system testing, and user acceptance testing (UAT). Unit testing examines individual modules or units to identify defects at the smallest software level. Integration testing evaluates the interaction and interfaces between integrated modules. System testing evaluates the complete integrated software to determine whether it satisfies the specified functional and non-functional requirements. Finally, User Acceptance Testing (UAT) evaluates the software from the user's or customer's perspective in an environment that resembles its intended operational environment.

4.1.8.20 Verification and Validation in Agile model

In the Agile model, verification and validation are performed continuously throughout the development process rather than being postponed until the end of the project. Since Agile development is iterative and incremental, each iteration or sprint typically includes requirements clarification, design, implementation, testing, review, and feedback.

Verification focuses on checking whether the software is being developed correctly according to the agreed requirements and technical specifications. Common verification techniques include requirements review, user-story review, acceptance-criteria review, design review, code review, pair programming, static code analysis, and automated code-quality checks. For example, before developing a user story for an online banking application, the development team may review its acceptance criteria to ensure that the requirements are clear, complete, and testable. During implementation, another developer may review the code or use a static analysis tool to identify potential defects without executing the program.

Validation in Agile focuses on determining whether the developed increment actually satisfies user needs and intended business objectives. Since working software is produced regularly, validation can take place within every sprint. Common validation techniques include unit testing, integration testing, functional testing, regression testing, system testing, acceptance testing, exploratory testing, usability testing, perfor-

SGOU

mance testing, and automated testing. For example, if a sprint develops a fund-transfer feature for a banking application, unit testing can verify individual functions, integration testing can check communication between the account and transaction modules, and functional testing can confirm that the complete feature behaves according to the acceptance criteria. At the end of the sprint, the product increment can be demonstrated to stakeholders during a sprint review, where their feedback helps determine whether the feature satisfies their actual needs.

Thus, in Agile, verification and validation are continuous and closely integrated with development. Verification is mainly concerned with "Are we building the product correctly?", while validation addresses "Are we building the right product for the users?". Unlike the traditional Waterfall model, where testing is often concentrated after implementation, Agile teams repeatedly verify and validate small increments of the product throughout the development cycle. This continuous approach allows defects, misunderstandings, and changes in user expectations to be identified and addressed early.

Recap

- Requirements verification checks whether requirements are correct, complete, consistent, clear, and properly defined.
- Requirements validation determines whether the requirements accurately represent stakeholder needs and expectations.
- Verification and Validation should be treated as continuous activities throughout the development life cycle.
- Early verification helps identify errors, ambiguities, and inconsistencies before they cause costly rework.
- Stakeholder involvement is essential for ensuring that requirements reflect actual business and user needs.
- Requirements should be unambiguous and measurable so that their implementation can be evaluated objectively.
- Requirements traceability helps establish relationships between requirements, design elements, implementation, and testing activities.
- Regular requirements reviews and inspections help detect defects at an early stage.
- Acceptance criteria provide measurable conditions for determining whether a requirement has been satisfactorily implemented.
- Prototypes can help stakeholders understand proposed functionality and identify misunderstandings before full development.

SGOU

- Appropriate automation tools can support requirements analysis, consistency checking, traceability, and documentation.
- AI-supported tools can assist requirements-related activities, but human review remains important for interpretation and decision-making.
- Poor-quality requirements can lead to development delays, increased costs, rework, and unsatisfactory software products.
- Systematic verification and validation improve alignment between stakeholder expectations and the developed software.
- Effective requirements verification and validation provide a strong foundation for developing reliable software that satisfies its intended requirements.

Objective Type Questions

1. Which activity determines whether a software work product conforms to specified requirements and applicable criteria?
2. Which activity determines whether the developed software is suitable for its intended use?
3. What is the primary focus of verification in software development?
4. What is the primary focus of validation in software development?
5. Which verification technique involves a structured examination of a software work product by qualified reviewers?
6. Which technique involves explaining a software work product step by step to other team members for examination?
7. Which testing technique examines individual software components or modules?
8. Which testing technique evaluates the interaction and data exchange between integrated modules?
9. Which testing technique evaluates the complete integrated software system?
10. Which testing technique determines whether the completed software is acceptable to its intended users?
11. Which SDLC model explicitly connects development activities with corresponding testing activities?

SGOU

12. Which verification technique examines source code without executing the program?

13. Which SDLC model emphasises continuous verification and validation through iterative development?

14. Which Agile activity allows stakeholders to evaluate a completed software increment and provide feedback?

15. Which V&V activity primarily evaluates software by executing it under defined conditions?

Answers to Objective Type Questions

1. Verification.
2. Validation.
3. Conformance.
4. Suitability.
5. Inspection.
6. Walkthrough.
7. Unit Testing.
8. Integration Testing.
9. System Testing.
10. Acceptance Testing.
11. V-Model.
12. Static Analysis.
13. Agile.
14. Review.
15. Validation.

SGOU

Assignments

1. Explain the role of Verification and Validation (V&V) in the Software Development Life Cycle. Differentiate between verification and validation with suitable examples.

2. Describe the verification activities performed during the requirements, design, implementation, and testing phases of the software life cycle. Explain how these activities help in early defect detection.

3. Explain the major validation techniques used in software development, including unit testing, integration testing, system testing, and User Acceptance Testing (UAT).

4. Describe the V-Model of the Software Development Life Cycle. Explain how each development phase is associated with its corresponding verification or validation activity.

5. Explain the Waterfall Model of SDLC, including its major phases, advantages, and limitations. Discuss how Verification and Validation are performed in the Waterfall model.

6. Describe the Agile Software Development Methodology. Explain its major characteristics, types of Agile methodologies, advantages, and limitations.

7. Compare Verification and Validation activities in the Waterfall, V-Model, and Agile development approaches. Explain how the timing and nature of V&V activities differ among these models.

8. Discuss the limitations and challenges of Verification and Validation in software development. Explain why V&V cannot guarantee completely defect-free software and how appropriate V&V practices can reduce development risks.

SGOU

Suggested Reading

1. Sethi, R. (2023). Software engineering: Basic principles and best practices. Cambridge University Press.

2. O'Regan, G. (2022). Concise guide to software engineering: From fundamentals to application methods. Springer.

3. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). Essentials of software testing. Cambridge University Press.

4. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

5. Sommerville, I. (2016). Software engineering (10th ed.). Pearson

Reference

1. Jalote, P. (2025). A concise introduction to software engineering: With open source and GenAI (2nd ed.). Springer.

2. Wang, et al. (2024). Enhancing human-machine pair inspection with risk number and code inspection diagram. Software Quality Journal, 32, 939–959

3. Pressman, R. S., & Maxim, B. R. (2023). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

4. Pfleeger, S. L., & Atlee, J. M. (2020). Software engineering: Theory and practice (5th ed.). Pearson.

SGOU



IEEE 1012 and Integrity

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the basic concepts and purpose of Verification and Validation (V&V) in system development
- Describe the different V&V processes used for system, software, and hardware development
- Identify the role of integrity levels in V&V
- Explain the purpose and major contents of a Verification and Validation Plan (VVP)

Prerequisite

Understanding software quality, risk assessment, system requirements, and testing concepts is essential before studying this topic because modern software systems often perform critical functions in real-world environments. Learners need to understand that not all software failures have the same consequences, and the level of assurance required depends on the potential impact of failure. They should be familiar with verification and validation activities, software testing, defect detection, and the relationship between requirements and system performance. For example, in an online banking system, a minor delay in displaying an account statement may be inconvenient, whereas an incorrect transaction could result in significant financial loss.

Similarly, in a medical monitoring system, a software error that incorrectly displays a patient's condition could potentially affect medical decisions. Such examples demonstrate the need to assess software according to the consequences associated with its failure.

A clear understanding of risk, reliability, safety, and the criticality of software functions therefore provides the necessary foundation for determining appropriate levels of verification and validation. This background enables learners to understand how systematic assurance activities can be planned according to the importance and potential consequences of a software system.

SGOU

Keywords

Life Cycle Processes, Integrity Levels, V&V Processes, System V&V, Software V&V, Hardware V&V, V&V Plan

Discussion

The IEEE 1012 is a process standard that defines Verification and Validation (V&V) processes in terms of specific activities and tasks. It is designed to be applied throughout the entire life cycle of a product, covering system, software, and hardware elements. The standard is highly flexible and compatible with various life cycle models, though it acknowledges that not all models will utilise every process listed.

4.2.1 Purpose and Scope of the IEEE Standard for System, Software, and Hardware Verification and Validation

The IEEE Standard for System, Software, and Hardware Verification and Validation provides a common framework for performing Verification and Validation (V&V) throughout the life cycle of systems, software, and hardware. Its main purpose is to determine whether development products conform to their specified requirements and whether they satisfy their intended use and the needs of users or stakeholders. V&V activities provide objective evidence through methods such as analysis, evaluation, review, inspection, assessment, and testing.

The standard establishes a systematic framework of V&V processes, activities, and tasks that can be applied throughout the life cycle. It defines the V&V tasks, their required inputs and outputs, the minimum V&V tasks associated with different integrity levels, and the contents of a Verification and Validation Plan.

The scope covers system, software, and hardware life cycle processes. It applies across activities such as acquisition, supply, development, operation, maintenance, and retirement, and it can also be applied to products that are being reused, including legacy and commercial off-the-shelf products. Software within the scope includes firmware and microcode, while system, software, and hardware also include their associated documentation.

The standard considers V&V as a life-cycle-wide activity rather than something performed only at the end of development. Verification provides evidence that products conform to requirements, standards, practices, and established criteria, whereas validation provides evidence that the resulting product satisfies its intended use, system requirements, and user needs. These two processes are complementary and use the results of each other to strengthen the overall V&V process.

The scope also considers interactions between software, hardware, users, operators, the operational environment, data, procedures, facilities, and other systems. This systems

SGOU

perspective is important because failures may occur not only within an individual component but also at the interfaces between system elements. Therefore, V&V activities examine aspects such as correctness, completeness, accuracy, consistency, testability, security, hazards, risks, and system interactions.

The standard supports different levels of V&V rigour through an integrity-level approach. The required intensity and rigour of V&V activities can be adjusted according to the importance and characteristics of the system. Higher integrity requirements involve more extensive analysis, formal techniques, and recording procedures. It also provides minimum and optional V&V tasks so that the V&V effort can be tailored to project requirements.

The current revision also aligns the V&V framework with contemporary system and software life-cycle standards, including the newer editions of the system and software life-cycle process standards. It explicitly addresses current development environments and terminology reflected in areas such as Agile development, adaptive approaches, artificial intelligence, machine learning, software as a service, and related V&V considerations. These areas are reflected among the current standard's subject keywords and application areas.

Thus, the overall purpose of the standard is to ensure that V&V is planned, systematic, objective, traceable, and appropriate to the level of risk and integrity of the system. It helps organisations detect defects and anomalies early, provide evidence of compliance, assess product and process quality, support risk management, and improve the reliability and suitability of systems, software, and hardware throughout their complete life cycle.

In simple terms: Verification asks “Was the product built correctly according to its requirements?”, while validation asks “Does the product meet its intended purpose and user needs?” The standard provides the structured processes and tasks needed to answer both questions throughout the life cycle. The current edition remains an active revision of the standard and retains this fundamental purpose and scope.

SGOU

4.2.2 Clauses of the IEEE Standard for System, Software, and Hardware Verification and Validation

The current IEEE Standards Association listing identifies the revised edition as an active standard and states that it continues to address system, software, and hardware V&V, their interfaces, different integrity levels, and V&V activities such as analysis, evaluation, review, inspection, assessment, and testing.

4.2.2.1 Clause 1: Overview

Clause 1 establishes the scope, purpose, application, objectives, organisation, audience, conformance, and general conditions for applying the standard. V&V is intended to operate throughout the life cycle rather than being treated only as a final testing activity. Verification provides evidence that a product is built according to requirements, while validation provides evidence that the product addresses the intended problem and user needs. For example, verifying a student management application includes checking

whether the specified login and registration requirements have been implemented correctly, whereas validation considers whether the application actually supports the activities expected by students and administrators.

The clause also explains that V&V should consider interactions among software, hardware, users, operators, data, procedures, facilities, and the operating environment. Thus, testing a software component in isolation is not always sufficient; its interaction with the hardware and other system components must also be considered.

4.2.2.2 Clause 2: Normative References

Clause 2 identifies the normative reference requirements. In the uploaded document, no normative references are prescribed by this clause, while useful standards are identified in the bibliography.

4.2.2.3 Clause 3: Definitions and Acronyms

Clause 3 establishes the terminology, definitions, abbreviations, and conventions needed to understand and apply the V&V requirements consistently. For example, terms such as verification, validation, acceptance testing, integrity level, and independent verification and validation have defined meanings within the standard. Consistent terminology is important because verification and validation are related but serve different purposes.

4.2.2.4 Clause 4: Relationship Between V&V and Life Cycle Processes

Clause 4 explains how V&V is connected with system, software, and hardware life cycle processes. Common V&V activities are used together with the appropriate system, software, or hardware V&V processes. The approach can also be applied recursively, from a complete system to its subsystems and then to software or hardware elements. For example, a hospital information system may require system-level V&V, while its patient registration software and associated hardware require their respective V&V activities.

4.2.2.5 Clause 5: Integrity Levels

Clause 5 uses integrity levels to determine the required scope, intensity, rigour, and depth of V&V. The standard defines four integrity levels, with higher integrity requiring more extensive and rigorous V&V. For a higher integrity application, V&V may need to consider abnormal conditions, boundary conditions, and comprehensive fault and recovery situations rather than only normal operation. The minimum V&V tasks are selected according to the assigned integrity level.

4.2.2.6 Clause 6: V&V Process Overview

Clause 6 explains how V&V activities and tasks are applied to the life cycle processes used by a project. Common activities are addressed for system, software, and hardware V&V, while specific activities are provided for each area. V&V testing is planned and performed as part of the life cycle. For example, testing should be planned when

SGOU

requirements and designs are developed so that appropriate test cases and procedures are available when the product reaches implementation and integration.

4.2.2.7 Clause 7: Common V&V Processes

Clause 7 contains V&V activities applicable across system, software, and hardware V&V. These include V&V management, acquisition support, supply planning, project planning, and configuration management. V&V management includes preparation and maintenance of the V&V Plan, review of V&V results, assessment of proposed changes, coordination with other life cycle activities, and identification of process improvement opportunities. For example, if a requirement is changed during development, V&V management determines which completed V&V tasks are affected and whether those tasks need to be repeated.

4.2.2.8 Clause 8: System V&V Processes

Clause 8 addresses V&V at the system level. The activities extend across business or mission analysis, stakeholder needs and requirements, system requirements, architecture, design, system analysis, implementation, integration, verification, transition, validation, operation, maintenance, and disposal. The purpose is to establish evidence that the system continues to satisfy requirements and intended use throughout its life cycle. For example, before a software and hardware system is transitioned into operational use, system V&V considers whether the integrated system satisfies its requirements and whether installation and operational conditions support its intended use.

4.2.2.9 Clause 9: Software V&V Processes

Clause 9 applies V&V to the software life cycle, covering software concept, requirements analysis, design, construction, integration, qualification testing, acceptance testing, verification, installation and checkout, validation, operation, maintenance, and disposal. This ensures that V&V is not limited to executing the final software test. For example, during requirements analysis, requirements can be evaluated for correctness and testability; during construction, source code and implementation can be evaluated; during integration, interfaces can be examined; and during operation, changes and operating procedures can be assessed.

4.2.2.10 Clause 10: Hardware V&V Processes

Clause 10 provides corresponding V&V processes for hardware. These cover hardware concept, requirements analysis, design, fabrication, integration, qualification testing, acceptance testing, verification, transition, validation, operation, maintenance, and disposal. For example, a newly developed electronic controller can be evaluated against its hardware requirements during design and fabrication, tested during qualification, checked during installation, and subsequently evaluated when operational conditions or maintenance activities change.

SGOU

4.2.2.11 Clause 11: V&V Reporting, Administration, and Documentation

Clause 11 establishes requirements for recording and communicating V&V results. It covers V&V reports, administrative requirements, and test documentation. Test documentation includes test plans, test designs, test cases, test procedures, and test results. Administrative arrangements include anomaly resolution and reporting, task iteration, deviations, control procedures, and applicable standards and practices. For example, when testing identifies a failure, the anomaly should be recorded, communicated, investigated, and tracked according to the defined V&V arrangements rather than being treated as an undocumented observation.

4.2.2.12 Clause 12: V&V Plan

Clause 12 defines the content of the Verification and Validation Plan (VVP). The plan provides the practical framework for conducting and controlling the V&V effort. Its content includes the purpose, referenced documents, definitions, V&V overview, organisation, schedule, integrity level, resources, responsibilities, tools and techniques, applicable V&V processes and tasks, reporting requirements, administrative requirements, and test documentation requirements. For example, a software project VVP can identify which requirements will be verified, which validation activities will be performed, who will conduct them, when they will occur, how anomalies will be handled, and what test evidence will be retained.

The clauses should be viewed as a connected V&V framework rather than as isolated topics. Clauses 1 to 6 establish the scope, terminology, life cycle relationship, integrity approach, and general V&V process. Clause 7 provides activities common to all V&V efforts, while Clauses 8, 9, and 10 address system, software, and hardware V&V respectively. Clauses 11 and 12 ensure that the V&V work is properly reported, documented, controlled, and planned. The standard therefore promotes continuous evidence-based assessment throughout the life cycle instead of relying only on final product testing.

4.2.3 Advantages of IEEE 1012

Advantages of the IEEE Standard for System, Software, and Hardware Verification and Validation

The standard provides a systematic and structured approach to Verification and Validation (V&V) throughout the life cycle of systems, software, and hardware. It establishes common V&V processes, activities, and tasks, making the V&V effort more organised and consistent.

1. Provides a common V&V framework

The standard establishes a common framework for V&V processes, activities, and tasks. It also specifies the required inputs and outputs for V&V activities, helping organisations conduct verification and validation in a consistent manner.

SGOU

2. Covers the complete life cycle

V&V is considered throughout the system, software, and hardware life cycle rather than only during final testing. This allows problems to be identified earlier during requirements, design, implementation, integration, operation, maintenance, and other life-cycle activities. The current revision continues this life-cycle approach across system, software, and hardware processes.

3. Supports different integrity levels

The standard uses integrity levels to determine the required V&V activities, tasks, rigour, and intensity. More critical systems can therefore receive more rigorous V&V activities than less critical systems.

4. Provides minimum V&V requirements

Minimum V&V tasks are specified according to the assigned integrity level. This provides a baseline that organisations can use to ensure that essential verification and validation activities are not omitted.

5. Allows tailoring to project requirements

In addition to minimum tasks, optional V&V tasks can be selected according to project requirements. This makes the V&V process adaptable to the characteristics, risks, complexity, and needs of a particular project.

6. Supports objective evidence

The standard emphasises analysis, evaluation, review, inspection, assessment, and testing. These activities provide objective evidence about whether products satisfy their requirements and intended use.

7. Supports early identification of problems

V&V activities performed throughout the life cycle can help identify defects, inconsistencies, incomplete requirements, and other problems before they become more expensive to correct.

8. Covers systems, software, hardware, and interfaces

The standard is not restricted to software alone. It covers systems, software, hardware, their interfaces, and related documentation. It can also be applied to developed, maintained, and reused items, including legacy and COTS components.

9. Supports risk and criticality considerations

Integrity determination can consider factors such as complexity, criticality, risk, safety, security, performance, and reliability. This helps organisations determine the appropriate intensity of V&V activities.

SGOU

10. Supports independent V&V

The standard provides consideration for the degree of independence required for V&V. This can help provide a more objective assessment, particularly for systems where independent assessment is important.

11. Supports different life-cycle models

The standard is designed to be compatible with different system, software, and hardware life-cycle models. The current revision continues this compatibility and covers a broader set of contemporary life-cycle applications.

12. Relevant to modern technology environments

The current revision reflects contemporary application areas such as adaptive and Agile development, artificial intelligence, machine learning, SaaS, COTS, security analysis, hazard analysis, regression testing, and reuse. These areas are identified among the current standard's published subject keywords.

4.2.4 Limitations of IEEE 1012

Limitations of the IEEE Standard for System, Software, and Hardware Verification and Validation.

The following limitations are practical considerations arising from the structure and application of the standard. They should be understood as implementation limitations rather than shortcomings explicitly stated by IEEE.

1. Can require considerable effort

A complete V&V program involves activities such as analysis, evaluation, reviews, inspections, assessments, and testing. For large or critical systems, implementing these activities can require considerable time, personnel, tools, and resources.

2. Can increase project cost

Preparing V&V plans, conducting reviews and tests, maintaining documentation, generating reports, and producing objective evidence can increase the overall cost of system development.

3. Requires skilled personnel

Effective V&V requires personnel who understand requirements engineering, testing, system engineering, software engineering, hardware, risk, security, and other relevant technical areas. Poorly trained personnel may not obtain the intended benefits from the standard.

4. Requires proper integrity-level determination

The V&V effort depends on appropriate assignment of integrity levels. The standard states that integrity levels are used to determine the activities, tasks, rigour, and intensity of V&V. Therefore, an inappropriate integrity assessment can lead to either insufficient or unnecessarily intensive V&V.

SGOU

5. Documentation can be extensive

The standard specifies V&V inputs, outputs, reports, and V&V planning activities. For large projects, maintaining this evidence and documentation can become substantial.

6. May be difficult for small projects

A small software project may not have sufficient resources, personnel, or budget to implement the full range of V&V activities. Appropriate tailoring is therefore important.

7. Does not replace engineering judgment

The standard provides processes and tasks, but organisations still need to determine appropriate techniques, tools, test environments, evidence, and levels of rigour for their particular system.

8. Does not prescribe one specific testing technique

The standard identifies V&V activities and tasks, but it does not provide a single testing method that is appropriate for every type of system. Organisations may therefore need additional testing standards, tools, and techniques.

9. Integration with organisational processes can be complex

The standard interacts with other life-cycle processes and organisational responsibilities. In large projects involving developers, suppliers, quality assurance teams, project management, and independent V&V teams, clearly defining responsibilities and interfaces can be challenging. The source document specifically recognises that some V&V activities may involve multiple organisations.

10. Requires continuous coordination

Since V&V is performed alongside life-cycle processes, changes in requirements, design, implementation, configuration, or risk conditions may require corresponding changes to V&V activities and evidence.

11. May require additional standards and guidance

The standard provides the V&V framework, but organisations may need to use other standards, organisational procedures, domain-specific requirements, testing technologies, security practices, or regulatory requirements to implement a complete assurance process.

12. Modern technologies may require specialised V&V techniques

Although the current revision reflects areas such as AI, machine learning, Agile/adaptive development, SaaS, security analysis, and reuse, applying conventional V&V principles to rapidly changing or highly complex technologies can still require specialised techniques and domain expertise. The published subject coverage confirms these contemporary application areas.

In short, the major advantages are systematic V&V, life-cycle coverage, integrity-based tailoring, objective evidence, risk consideration, and applicability to systems, software,

SGOU

and hardware. The major practical limitations are the possible increase in cost, time, documentation, skill requirements, and organisational complexity.

The current revision remains focused on the same fundamental purpose of establishing a common V&V framework, defining V&V tasks and their inputs and outputs, identifying minimum tasks according to integrity levels, and defining the V&V plan.

Recap

- IEEE 1012 provides a structured Verification and Validation (V&V) framework for systems, software, and hardware.
- V&V is treated as a life cycle activity rather than being limited to final product testing.
- Verification determines whether a product conforms to its specified requirements.
- Validation determines whether the resulting product satisfies its intended use and user needs.
- V&V activities can include analysis, evaluation, review, inspection, assessment, and testing.
- The standard connects V&V activities with the relevant system, software, and hardware life cycle processes.
- V&V considers not only individual components but also interfaces and interactions among system elements.
- Integrity levels are used to determine the appropriate extent, intensity, and rigour of V&V activities.
- The framework identifies minimum V&V tasks associated with different integrity levels.
- V&V activities can be tailored according to the product characteristics, project requirements, risks, and integrity level.
- System V&V evaluates the complete system, including requirements, architecture, integration, interfaces, behaviour, and intended use.
- Software V&V covers software requirements, design, implementation, integration, testing, operation, maintenance, and applicable reuse activities.
- Hardware V&V evaluates hardware requirements, design, fabrication, integration, testing, operation, maintenance, and interfaces.

SGOU

- V&V documentation and reporting provide objective evidence, support traceability, and record V&V results, findings, anomalies, and corrective actions.
- A V&V Plan provides the basis for organising V&V activities, including responsibilities, schedules, resources, applicable tasks, methods, evidence, reporting, and test documentation.

Objective Type Questions

1. Which process determines whether a product conforms to specified requirements?
2. Which process determines whether a product satisfies its intended use and user needs?
3. Which activity involves examining a product or process systematically to obtain evidence?
4. How many integrity levels are defined in the standard?
5. Which document provides the framework for planning V&V activities?
6. Which type of software includes firmware and microcode within the scope of the standard?
7. What type of products includes previously developed products that can be reused?
8. Which V&V level considers the complete system and interactions among its elements?
9. Which V&V area focuses specifically on software requirements, design, implementation, and testing?
10. Which V&V area focuses specifically on hardware products and their interfaces?
11. Which activity involves examining software or system products to identify defects or inconsistencies?
12. What type of evidence should V&V activities produce?

SGOU

Answers to Objective Type Questions

1. Verification.
2. Validation.
3. Assessment.
4. Four.
5. V& VPlan
6. Software.
7. Legacy.
8. System.
9. Software.
10. Hardware.
11. Inspection.
12. Objective.

Assignments

1. Explain the purpose and scope of the Verification and Validation process. Describe how V&V activities are applied throughout the complete system, software, and hardware life cycle.
2. Explain the concept of integrity levels in Verification and Validation. How do integrity levels influence the extent, rigour, and selection of V&V tasks? Give suitable examples.
3. Describe the major Verification and Validation processes for system, software, and hardware. Explain the importance of verifying interfaces and interactions among these components.
4. Explain the purpose and major contents of a V&V Plan. Discuss the advantages and limitations of applying a structured V&V process in software and system development.

SGOU

Reference

1. IEEE Std 1012, IEEE Standard for System, Software, and Hardware Verification and Validation.

2. Jalote, P. (2025). A concise introduction to software engineering: With open source and GenAI (2nd ed.). Springer.

3. Wang, et al. (2024). Enhancing human-machine pair inspection with risk number and code inspection diagram. Software Quality Journal, 32, 939–959

4. Pressman, R. S., & Maxim, B. R. (2023). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

5. Pfleeger, S. L., & Atlee, J. M. (2020). Software engineering: Theory and practice (5th ed.). Pearson.

Suggested Reading

1. IEEE Std 1012, IEEE Standard for System, Software, and Hardware Verification and Validation.

2. Sethi, R. (2023). Software engineering: Basic principles and best practices. Cambridge University Press.

3. O'Regan, G. (2022). Concise guide to software engineering: From fundamentals to application methods. Springer.

4. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). Essentials of software testing. Cambridge University Press.

5. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

6. Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

SGOU



V&V for Requirements

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the meaning and importance of requirements verification and validation.
- differentiate between requirements verification and requirements validation.
- describe the major techniques used for verifying and validating requirements.
- apply appropriate validation practices to identify problems in software requirements.

Prerequisite

The learners should understand the basic concepts of software requirements and the role they play in software development. They should be familiar with how requirements are collected, documented, analysed, and communicated to developers and stakeholders. A basic understanding of software development life cycle stages is also useful for understanding when requirements are examined and evaluated.

Learners should know that unclear, incomplete, or conflicting requirements can lead to defects, delays, and increased development costs. For example, when developing an online banking system, a requirement such as "the system should provide secure transactions" needs to be clearly defined so that security expectations can be understood and evaluated.

Learners should also have a basic awareness of techniques such as reviews, inspections, traceability, and test planning that can be used to assess requirements. Understanding the difference between checking whether requirements are correctly specified and determining whether they actually represent user needs will help learners appreciate the purpose of these activities. This foundation enables learners to understand how systematic examination of requirements can reduce errors early and improve the likelihood of developing software that meets its intended purpose.

SGOU

Keywords

Requirements Traceability, Prototyping, Acceptance Criteria, Stakeholders, Ambiguity, Completeness

Discussion

In today's rapidly changing software development environment, **requirements verification and validation** play an important role in ensuring the quality and success of a project. These activities help identify errors and inconsistencies in requirements, ensure that stakeholder expectations are properly addressed, and confirm that the requirements comply with relevant standards and project objectives. By verifying the correctness, completeness, and consistency of requirements and validating them against actual user needs, organisations can reduce development risks, minimise rework, and improve the quality of the final software product.

A clear understanding of requirements verification and validation is therefore essential for effective software development. These activities involve appropriate review techniques, validation practices, and supporting tools that help ensure that requirements are properly defined and understood before and during development. Applying these practices systematically enables project managers, business analysts, developers, and other stakeholders to establish a reliable foundation for developing software that satisfies both functional requirements and stakeholder expectations.

4.3.1 Requirements Verification

Requirements Verification is the process of systematically examining software requirements to determine whether they are **complete, correct, consistent, clear, and properly specified**. It ensures that the requirements document contains all the necessary information needed to guide subsequent software development activities and provides a reliable basis for design and implementation.

Requirements should be verified and approved before the design phase begins, because errors or omissions identified at an early stage can be corrected with comparatively less effort. If requirements are not properly examined before development starts, incorrect, incomplete, or ambiguous requirements may remain unnoticed and lead to problems during design, implementation, and testing. For example, if a requirement for an online admission system does not specify how incomplete applications should be handled, developers and testers may interpret the requirement differently, resulting in rework later. Therefore, requirements verification should be performed **early and repeatedly** throughout the requirements engineering process to identify and resolve problems before they affect subsequent development activities.

SGOU

4.3.2 Importance of Requirements Verification

The primary objectives of requirements verification are to ensure that the system requirements are **complete, correct, and consistent**. Verification helps determine whether all necessary requirements have been properly identified, accurately documented, and expressed without contradictions or ambiguities.

Conducting verification at an early stage helps identify **missing, incorrect, or invalid requirements** before they affect subsequent development activities. Detecting and correcting such issues early can significantly reduce rework, development effort, and additional costs. Resolving a requirement-related problem at the beginning is generally much easier than discovering the same problem later, after substantial design and coding work has been completed.

Requirements verification is therefore essential for establishing a reliable foundation for software development. **Incomplete, incorrect, or inconsistent requirements can create difficulties during design, implementation, testing, and deployment**, and may ultimately prevent the developed system from achieving its intended objectives and expected functionality.

4.3.3 Requirement Validation

Validation is the process of determining whether the developed system satisfies the intended needs, requirements, and expectations of its users and stakeholders. It addresses the question, **"Are we building the right system?"** by evaluating whether the completed system performs the intended functions and is suitable for its purpose. Validation may involve techniques such as **black-box testing, white-box testing, integration testing, and unit testing** to assess whether the system behaves as expected. Requirements validation is also performed during the early stages of development to identify incorrect, incomplete, ambiguous, or inappropriate requirements before they lead to significant rework. Therefore, validating requirements at an early stage helps ensure that they are appropriate, accurate, and consistent with the intended objectives and applicable standards.

4.3.4 Importance of Requirements Validation

Validating requirements helps identify problems or inconsistencies in the requirements gathered during earlier stages of requirements engineering. Conducting validation during the early stages of the development life cycle makes it possible to detect errors before they become difficult and costly to correct. Early validation improves the accuracy, clarity, and completeness of the collected requirements by identifying and addressing potential defects. Without proper validation, incorrect or unclear requirements may lead to inappropriate development decisions and inaccurate outcomes. Another important benefit of requirements validation is the reduction of maintenance costs, as resolving problems at an early stage generally requires less effort than correcting them after development. A well-validated set of requirements therefore provides a strong foundation for the software project, supports a more reliable development process, and reduces the possibility of failures, rework, and rejection.

SGOU

4.3.5 Difference Between Requirements Verification and Validation

Verification and validation are two closely related but distinct activities in software development, and they are often confused with one another. Although both are concerned with ensuring software quality, they address different aspects of the development process.

Validation focuses on determining whether a product, service, or system satisfies the needs and expectations of the customer and other identified stakeholders. It is concerned with establishing whether the developed system is suitable for its intended purpose and acceptable to its users. In other words, validation asks whether the right system has been developed.

Verification, on the other hand, focuses on determining whether a product, service, or system conforms to specified requirements, standards, regulations, specifications, or other defined conditions. It examines whether the development work has been carried out correctly according to the established requirements. In simple terms, verification asks whether the system has been developed correctly.

In the context of requirements engineering, requirements verification involves examining the requirements to determine whether they are complete, correct, consistent, clear, and properly specified. Requirements validation focuses on determining whether those requirements accurately represent the actual needs and expectations of the users and stakeholders. Thus, verification primarily checks conformance to specified requirements, whereas validation focuses on fitness for purpose and satisfaction of stakeholder needs.

4.3.6 Techniques Used in Requirements Verification

There are various tools and techniques that can be used in Requirements Verification, including inspections, demonstrations, and tests.

- **Inspections:** Inspections are reviews of system requirements that are conducted by a team of experts. The purpose of an inspection is to identify errors, omissions, or inconsistencies in the requirements document.
- **Demonstrations:** Demonstrations involve demonstrating the functionality of the system to stakeholders. This is typically done using prototypes or software simulations.
- **Tests:** Tests are used to verify that the system meets its functional requirements. Functional testing includes black-box testing, white-box testing, and regression testing.

4.3.7 When to Validate?

Requirements validation is a **continuous activity** carried out throughout the requirements engineering process to ensure that stakeholder, solution, and transition requirements remain aligned with the identified business requirements. Rather than

SGOU

treating validation as a single activity performed at the end of requirements engineering, it should be performed at every relevant stage.

During requirements elicitation, the identified requirements should be reviewed and cross-checked against the original sources from which they were gathered to ensure that important information has not been misunderstood or omitted. During requirements analysis and negotiation, the proposed requirements should be examined to determine whether they are appropriate, relevant, and acceptable to the stakeholders. The resulting requirements document should then be reviewed to confirm that it represents the agreed requirements accurately. During requirements specification, the documented requirements should be compared with the actual needs and expectations of the users. They should also be checked against applicable rules, standards, and established requirements criteria. Continuous validation in this manner helps identify problems early and ensures that the requirements remain appropriate throughout the requirements engineering process.

4.3.8 Requirements Validation Techniques

Several techniques can be applied to validate software requirements and ensure that they are clear, complete, consistent, traceable, and aligned with the intended needs. Requirement Checks involve carefully examining and proofreading the requirements documentation to confirm that important information gathered during requirements elicitation has not been omitted. The requirements are also examined for traceability by establishing links between individual requirements and their related sources, design elements, or other requirements. A Requirements Traceability Matrix (RTM) can be used for this purpose to ensure that each requirement is properly accounted for and justified. The requirements are also checked for clarity, consistency, completeness, and appropriate documentation format.

- Prototyping involves developing a preliminary model or simulation of the proposed system to provide users and stakeholders with a better understanding of how the system may function. The prototype allows stakeholders to interact with or visualise important aspects of the proposed system and provide feedback. For example, a prototype of a university admission system may allow admission staff to view the proposed application submission and document-verification screens before the complete system is developed. Feedback obtained from users can help identify unclear, missing, or inappropriate requirements at an early stage.

- Test Design involves developing test scenarios based on the specified requirements to determine whether the requirements can be evaluated effectively. Functional requirements can generally be associated with corresponding test cases so that each requirement can be checked through appropriate testing. Non-functional requirements may require more careful test planning because their verification may involve specific performance, security, usability, or other measurable conditions. Designing tests from requirements can therefore help reveal ambiguous requirements, missing details, or requirements that cannot be evaluated effectively.

SGOU

- Requirements Review is a structured examination of the requirements by a group of knowledgeable stakeholders, analysts, developers, testers, or other relevant participants. During the review, the requirements are systematically examined to identify potential errors, omissions, inconsistencies, ambiguities, and other problems. The identified issues are then discussed, and appropriate corrective actions are determined. A review checklist containing relevant quality criteria can be used to make the examination systematic and consistent. Once the identified issues

4.3.9 Principles of Requirements Validation

Considering the following **six principles of requirements validation** can help improve the effectiveness and quality of the validation process:

Principle 1: Involve the Appropriate Stakeholders

Ensure that the right stakeholders participate in the validation process so that requirements are reviewed from relevant perspectives.

Principle 2: Separate Error Identification from Error Correction

Identify and document problems in the requirements before deciding how those problems should be corrected. This helps ensure that errors are systematically recognised and evaluated.

Principle 3: Validate from Multiple Perspectives

Examine requirements from different viewpoints to identify inconsistencies, omissions, ambiguities, and other potential problems that may not be visible from a single perspective.

Principle 4: Use Appropriate Documentation Formats

Review requirements using suitable forms or types of documentation so that different aspects of the requirements can be examined effectively.

Principle 5: Develop Supporting Development Artefacts

Create relevant development artefacts, such as models, prototypes, or other representations, to support the examination and validation of requirements.

Principle 6: Perform Validation Repeatedly

Requirements validation should be carried out repeatedly throughout the development process rather than being treated as a one-time activity. This helps ensure that requirements remain appropriate as the system evolves.

4.3.10 Requisites for Requirements Validation

- **Requirements Document:** The requirements document should be complete rather than an unfinished draft. It should be properly structured, formatted, and organised in accordance with the applicable organisational standards.

- **Organisational Knowledge:** Relevant knowledge about the organisation, including implicit or previously established knowledge, should be available to assess whether

SGOU

the proposed requirements are realistic and practical.

- **Organisational Standards:** The applicable standards and guidelines of the organisation should be available, particularly those governing the preparation, structure, format, and presentation of the requirements document.

4.3.11 Requirements Validation Outputs

The outputs of requirements validation provide a formal record of the issues identified during the validation process, and the actions agreed upon to address those issues. These outputs help ensure that problems found in the requirements are properly documented, discussed, and followed up.

4.3.11.1 Problem List

A problem list is a documented record of the problems identified during the examination and validation of the requirements document. The identified problems may include unclear, incomplete, inconsistent, incorrect, or missing requirements. Recording these issues systematically helps the project team understand what needs to be reviewed or corrected. For example, if a requirement states that "the system should process user requests quickly" without specifying what "quickly" means, this ambiguity can be recorded in the problem list for further clarification.

4.3.11.2 Agreed Actions

Agreed actions describe the corrective or follow-up activities decided upon to address the problems identified in the requirements document. Each identified problem is reviewed by the relevant stakeholders, and appropriate actions are determined. A single problem may require more than one corrective action. For example, an unclear requirement may require consultation with the customer, modification of the requirement statement, and subsequent review by the requirements team. On the other hand, some identified problems may not require any corrective action if the stakeholders determine that the issue does not need to be changed. Therefore, the agreed actions provide a clear record of how identified requirements problems are to be handled.

4.3.12 Best Practices for Requirements Validation and Verification

Effective requirements Verification and Validation (V&V) are essential for reducing project risks, ensuring that requirements are properly understood, and improving the quality of the final software product. These activities should not be treated as isolated tasks performed only at the end of requirements engineering. Instead, they should be carried out systematically throughout the development process so that errors, omissions, ambiguities, and inconsistencies can be identified and addressed as early as possible.

4.3.12.1 Involve Stakeholders Early and Continuously

All relevant stakeholders, including customers, end users, project managers, developers, testers, and technical specialists, should be involved in requirements verification and

SGOU

validation from the early stages of the project. Their participation helps ensure that the documented requirements accurately represent actual business and user needs. Early involvement also provides opportunities to identify misunderstandings before they become costly problems.

For example, when developing a university admission management system, admission officers, administrators, students, developers, and testing teams can review the requirements together. Admission officers can explain the actual admission procedures, while technical teams can identify requirements that may be difficult to implement. This collaborative approach helps produce requirements that are both useful to users and technically feasible.

4.3.12.2 Automate Verification and Validation Activities Where Appropriate

Automation can be used to support repetitive verification and validation activities. Automated tools can assist in activities such as checking requirements for consistency, identifying duplicate requirements, monitoring traceability, mapping requirements to test cases, and checking compliance with predefined rules. Automation can reduce manual effort and help minimise errors that may occur during repetitive activities.

For example, if a project contains hundreds of requirements, manually checking whether every requirement has an associated test case can be difficult. A requirements management tool can automatically identify requirements that have not been linked to corresponding test cases, allowing the verification team to investigate the gaps more efficiently.

4.3.12.3 Maintain Requirements Traceability

Requirements traceability should be maintained throughout the software development life cycle. A **Requirements Traceability Matrix (RTM)** can be used to establish relationships between requirements and related design elements, implementation components, and test cases. Effective traceability helps determine whether every requirement has been appropriately considered and whether the implemented system can be traced back to its original requirements.

For example, if a requirement states that an admission system must generate an admission confirmation receipt, the requirement can be linked to the relevant design component, implementation module, and test case. If the requirement changes later, these relationships help identify the associated components that may also need to be reviewed or modified.

4.3.12.4 Conduct Regular Reviews and Revalidation

Requirements should be reviewed regularly rather than being considered final immediately after they are documented. Changes in business needs, user expectations, technology, regulations, or project constraints may make previously defined requirements unsuitable. Regular reviews allow the development team and stakeholders to determine whether the requirements remain correct, relevant, complete, and consistent.

SGOU

For example, if a university introduces a new online payment facility during the development of an admission system, the existing requirements related to payment processing may need to be reviewed. The new requirement may also affect security, database, interface, testing, and reporting requirements. Regular revalidation helps identify such effects before implementation progresses too far.

4.3.12.5 Use AI-Supported Tools Carefully

AI-supported requirements management and validation tools can assist teams in identifying potential ambiguities, inconsistencies, missing information, duplicate requirements, or relationships between requirements. Such tools can also support activities such as requirement classification, traceability analysis, and comparison against predefined rules or standards. AI-based results should be treated as supporting information rather than a replacement for human judgement. For example, an AI-based tool may identify a requirement as potentially ambiguous, but an experienced analyst or stakeholder should determine whether the requirement actually needs clarification and what wording would accurately represent the intended need.

4.3.12.6 Define Clear and Measurable Acceptance Criteria

Requirements should be expressed in a manner that allows their achievement to be determined objectively. Wherever appropriate, measurable acceptance criteria should be associated with individual requirements. Clear criteria make it easier for reviewers, developers, and testers to determine whether the requirement has been correctly implemented.

For example, instead of stating that "the system should respond quickly," a requirement can define an appropriate measurable response-time criterion. This makes the requirement easier to verify and provides testers with a clear basis for determining whether the expected behaviour has been achieved.

4.3.12.7 Maintain Complete and Organised Documentation

All important verification and validation activities should be properly documented. Records may include review findings, identified problems, corrective actions, approval decisions, requirement changes, traceability information, and testing results. Proper documentation provides evidence of what was reviewed, what problems were identified, and how those problems were addressed.

For example, during a requirements review, a reviewer may identify an incomplete requirement. The issue can be recorded along with the proposed correction, responsible person, review status, and final approval. Such records provide useful evidence for future reviews and help maintain consistency throughout the project.

4.3.12.8 Use Prototypes and Simulations When Appropriate

Prototypes and simulations can be useful for validating requirements, particularly when users may find it difficult to understand a requirement from textual descriptions alone. A prototype provides stakeholders with a representation of the proposed system and

SGOU

allows them to evaluate whether the expected functionality and user interaction are appropriate.

For example, before developing the complete interface of an online learning system, a prototype can be presented to teachers and students. They can evaluate the navigation, course registration, assignment submission, and other proposed functions. Their feedback can then be used to clarify or modify the requirements before substantial development work is undertaken.

Requirements verification and validation should be treated as continuous and structured activities throughout the development process rather than as a single final-stage exercise. Requirements verification systematically examines whether the specified requirements are complete, correct, consistent, unambiguous, and properly defined, while requirements validation determines whether they accurately reflect the needs and expectations of stakeholders.

Conducting both activities early and regularly, with active stakeholder involvement, appropriate automation, requirements traceability, regular reviews, measurable acceptance criteria, comprehensive documentation, and the use of prototypes, can help identify errors and inconsistencies before they result in costly rework or development delays.

AI-supported tools can further assist in reviewing and analysing requirements when used appropriately, while human review remains essential for interpreting requirements and making informed decisions. Applying these practices systematically provides a reliable foundation for system development, improves alignment between stakeholder expectations and the software product, reduces unnecessary rework, and increases confidence that the developed system satisfies its intended requirements.

Recap

- Requirements verification checks whether requirements are complete, correct, consistent, and clearly specified.
- Verification helps identify errors and omissions before the design and implementation stages.
- Requirements validation checks whether the requirements represent the actual needs and expectations of stakeholders.
- Verification asks, “Are we developing the system correctly?”
- Validation asks, “Are we developing the right system?”
- Verification and validation should be performed early and repeatedly throughout the requirements engineering process.
- Requirements reviews and inspections help identify ambiguities, inconsistencies, omissions, and incorrect requirements.

SGOU

- Prototyping helps stakeholders understand the proposed system and provide feedback on requirements.
- Requirements Traceability Matrix (RTM) helps establish relationships between requirements and related development and testing activities.
- Test design can help determine whether requirements are clear, measurable, and testable.
- Requirements validation should involve appropriate stakeholders from different perspectives.
- A problem list records issues identified during requirements validation.
- Agreed actions document the corrective or follow-up activities decided for identified problems.
- Clear and measurable acceptance criteria make requirements easier to verify and validate.
- Continuous verification and validation can reduce rework, development risks, and maintenance costs while improving confidence in the final software product.

Objective Type Questions

1. What is the process of systematically examining software requirements called?
2. Which activity determines whether requirements are complete, correct, consistent, and clear?
3. Which activity determines whether requirements represent the actual needs of stakeholders?
4. What document is used to establish relationships between requirements and related development artefacts?
5. Which technique develops a preliminary model of the proposed system?
6. What technique involves a structured examination of requirements by knowledgeable stakeholders?
7. What is the documented record of problems identified during requirements validation called?

SGOU

8. What records the corrective actions decided for identified requirements problems?

9. Which technique involves developing test scenarios based on requirements?

10. What should be maintained between requirements and design, implementation, and testing artefacts?

11. Which activity should be performed repeatedly throughout the requirements engineering process?

12. What type of criteria make requirements easier to evaluate objectively?

13. Which type of tools can assist in identifying ambiguities and inconsistencies in requirements?

14. What should remain essential when using AI-supported requirements validation tools?

Answers to Objective Type Questions

1. Verification.
2. Verification.
3. Validation.
4. RTM.
5. Prototyping.
6. Review.
7. Problem-list.
8. Action log
9. Test case generation
10. Traceability.
11. Validation.
12. Measurable.
13. AI.
14. Judgement.

SGOU

Assignments

1. Explain the concept of Requirements Verification in detail. Discuss its objectives and explain why requirements should be verified before the design phase begins, with a suitable example.

2. Describe the importance of Requirements Verification in software development. Explain how identifying incomplete, incorrect, or inconsistent requirements at an early stage can reduce development effort, cost, and rework.

3. Explain Requirements Validation in detail. Discuss its purpose, importance, and role in determining whether the requirements and developed system satisfy the needs and expectations of users and stakeholders.

4. Differentiate between Requirements Verification and Requirements Validation. Explain the difference between conformance to specified requirements and fitness for purpose using suitable examples.

5. Explain the major techniques used in Requirements Verification and Validation, including inspections, demonstrations, tests, requirement checks, prototyping, test design, and requirements review. Discuss the usefulness of each technique with examples.

6. Explain why Requirements Validation is considered a continuous activity. Describe the validation activities that can be performed during requirements elicitation, analysis and negotiation, and specification stages.

7. Discuss the six principles of Requirements Validation. Explain the importance of stakeholder involvement, separating error identification from correction, multiple perspectives, suitable documentation formats, supporting development artefacts, and repeated validation.

8. Explain the best practices for Requirements Verification and Validation. Discuss stakeholder involvement, automation, requirements traceability, regular reviews, careful use of AI-supported tools, measurable acceptance criteria, organised documentation, and the use of prototypes and simulations.

SGOU

Suggested Reading

1. Sethi, R. (2023). Software engineering: Basic principles and best practices. Cambridge University Press.

2. O'Regan, G. (2022). Concise guide to software engineering: From fundamentals to application methods. Springer.

3. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). Essentials of software testing. Cambridge University Press.

4. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

5. Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Reference

1. Jalote, P. (2025). A concise introduction to software engineering: With open source and GenAI (2nd ed.). Springer.

2. Wang, et al. (2024). Enhancing human-machine pair inspection with risk number and code inspection diagram. Software Quality Journal, 32, 939–959

3. Pressman, R. S., & Maxim, B. R. (2023). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

4. Pfleeger, S. L., & Atlee, J. M. (2020). Software engineering: Theory and practice (5th ed.). Pearson.

SGOU



Software Standards and Validation

Learning Outcomes

After completion of this unit, learner will be able to:

- summarise the key features of ISO/IEC/IEEE 12207:2026
- identify the different categories of software life cycle processes
- compare verification and validation activities within the software life cycle
- interpret the maturity and capability levels of CMMI

Prerequisite

Before studying structured software development and process improvement frameworks, learners need to understand the importance of well-defined requirements, systematic development processes, quality management, documentation, project planning, and continuous process improvement in software engineering. A clear understanding of the software development life cycle (SDLC) helps learners recognise how activities such as requirements analysis, design, implementation, testing, deployment, and maintenance are organised.

Learners should also understand why organisations need consistent processes to ensure that software is developed systematically and meets customer expectations. Knowledge of verification and validation is important for understanding how work products and completed software can be evaluated for correctness, quality, and suitability.

An understanding of project management concepts, including planning, scheduling, resource management, risk management, and monitoring, provides a foundation for studying process-oriented approaches.

Learners should also be familiar with the importance of process measurement and performance evaluation for identifying weaknesses and improving software development practices. For example, when a hospital develops a patient management system, clearly defined requirements, documented processes, systematic testing, quality checks, and continuous monitoring can help prevent errors that could affect patient services. Therefore, understanding these fundamental concepts is necessary to appreciate how structured frameworks help organisations establish disciplined processes, improve software quality, manage development activities effectively, and achieve more predictable outcomes.

SGOU

Keywords

Process Framework, Stakeholders, Process Maturity, Continuous Improvement, Quantitative Management

Discussion

4.4.1 ISO/IEC/IEEE 12207:2026 – Software Life Cycle Processes

Software systems are developed, operated, maintained, modified, and eventually retired within a life cycle. Managing these activities systematically is essential for producing software that satisfies stakeholder needs and achieves the intended business and technical objectives. ISO/IEC/IEEE 12207:2026, Systems and software engineering — Software life cycle processes, provides a common framework for defining and managing processes throughout the software life cycle. The standard is intended for software systems, products, and services and can be applied from conception and development through operation, maintenance, support, and retirement.

The standard does not prescribe one particular software development methodology or SDLC model. Instead, it provides a set of processes, activities, and tasks from which an organisation or project can select an appropriate combination according to its objectives and circumstances. Thus, the processes can be incorporated into different development approaches, including iterative and Agile approaches.

4.4.1.1 Purpose of ISO/IEC/IEEE 12207

The principal purpose of ISO/IEC/IEEE 12207 is to establish a **common process framework for the software life cycle**. The framework provides a structured way for organisations and projects to define, perform, manage, assess, and improve software life cycle processes.

The standard considers the interests of different participants in the software life cycle, including acquirers, suppliers, developers, integrators, operators, maintainers, managers, quality personnel, and users. It addresses stakeholder needs and constraints, requirements establishment, design, implementation, integration, verification, validation, operation, maintenance, and disposal.

For example, when an organisation develops a university admission system, the standard does not tell the organisation to use Waterfall, Agile, or another specific development model. Instead, it provides processes that can be incorporated into the selected development approach. The organisation can determine how these processes will be performed, who will perform them, what evidence will be produced, and how the results will be assessed.

SGOU

4.4.1.2 Scope and Applicability

ISO/IEC/IEEE 12207 applies to a wide range of software systems. It can be used for a one-of-a-kind software system, a commercially distributed product, a customised system, or software embedded within a larger system. It also applies to software that forms part of a system of systems.

The standard covers the complete software life cycle. This includes conception, development, operation, support, maintenance, and retirement. It can also be applied when software systems or services are acquired from external suppliers rather than developed internally. The processes may be performed concurrently, iteratively, recursively, or incrementally depending on the nature of the system and the chosen life cycle approach.

This makes the standard applicable to both traditional and modern software development environments.

4.4.1.3 Basic Concepts of the Standard

A central concept in ISO/IEC/IEEE 12207 is the **process**. A process consists of related activities that transform inputs into outputs. Each process has a defined **purpose** and expected **outcomes**. A process purpose describes the overall objective of performing the process, while process outcomes represent observable results that indicate successful implementation.

The standard therefore provides more than a list of software development activities. It establishes a process-oriented framework in which organisations can identify what needs to be achieved and then determine suitable activities and tasks for achieving those outcomes.

For example, in a software project, the implementation process may result in software elements that satisfy the specified implementation requirements. The integration process may then establish an integrated software product, while verification and validation processes provide evidence that the product has been appropriately checked and evaluated.

4.4.1.4 Life Cycle Process Framework

ISO/IEC/IEEE 12207:2026 organises its processes into major groups. The major process groups covered in the standard are:

1. Agreement processes
2. Organisational project-enabling processes
3. Technical management processes
4. Technical processes

The standard's contents identify the processes within the organisational project-enabling, technical management, and technical groups. The technical processes extend from

SGOU

business or mission analysis through stakeholder needs, requirements, architecture, design, implementation, integration, verification, transition, validation, operation, maintenance, and disposal.

These groups should not be interpreted as a rigid sequence. Software life cycle processes can operate concurrently, iteratively, and recursively. An organisation selects and applies an appropriate subset of the processes according to its particular objectives.

1. Agreement Processes

Agreement processes establish the relationship between parties involved in acquiring and supplying a software system, product, or service. They provide a framework through which the parties can determine what is to be acquired or supplied and establish appropriate responsibilities and commitments.

For example, if a university contracts a software company to develop an online learning management system, the university acts as the acquirer and the software company acts as the supplier. The agreement needs to establish what is expected from each party, including the required software product or service and the responsibilities associated with its development and delivery.

The standard supports the selection, negotiation, agreement, and performance of relevant processes and activities between acquirers and suppliers.

2. Organisational Project-Enabling Processes

Organisational project-enabling processes provide the environment and organisational capabilities needed to support software projects. The standard identifies processes such as infrastructure management, portfolio management, human resource management, quality management, and knowledge management within this area.

These processes are important because successful software development depends not only on programming activities but also on appropriate infrastructure, competent personnel, organisational knowledge, quality practices, and alignment with organisational objectives.

For example, a software organisation developing several applications may need appropriate development infrastructure, skilled personnel, quality management practices, and mechanisms for sharing lessons learned between projects.

3. Technical Management Processes

Technical management processes provide mechanisms for planning, monitoring, controlling, and managing technical work. The standard identifies project planning, project assessment and control, decision management, risk management, configuration management, information management, measurement, and quality assurance as technical management processes.

These processes operate throughout the life cycle rather than being restricted to a single development phase.

SGOU

For example, risk management may identify the possibility that an external payment service will not meet the required performance. Configuration management can control different versions of source code and other configuration items. Measurement can provide objective information about project progress or product characteristics, while quality assurance supports confidence that appropriate processes are being followed.

4. Technical Processes

Technical processes form a major part of ISO/IEC/IEEE 12207. The standard identifies the following technical processes:

Business or mission analysis → Stakeholder needs and requirements definition → System requirements definition → System architecture definition → Design definition → System analysis → Implementation → Integration → Verification → Transition → Validation → Operation → Maintenance → Disposal.

These processes collectively address the development and use of a software system throughout its life cycle.

4.4.1.5 Business or Mission Analysis Process

Business or mission analysis establishes an understanding of the problem, opportunity, business objectives, or mission that the system is intended to address.

For example, before developing an online university admission system, the organisation may analyse the existing admission process, identify problems faced by applicants and administrators, and determine the intended improvements. This provides the foundation for subsequent requirements and system definition activities.

4.4.1.6 Stakeholder Needs and Requirements Definition Process

This process establishes and defines the needs, expectations, concerns, priorities, and constraints of relevant stakeholders. Stakeholders may include users, customers, managers, operators, regulators, and other parties affected by the system.

For example, applicants may need an easy-to-use application interface, while admission officers may require facilities for verifying applications and generating reports. These different stakeholder perspectives need to be considered when establishing requirements.

4.4.1.7 System Requirements Definition Process

System requirements define what the system is expected to provide and the characteristics it needs to possess.

Requirements may describe functional capabilities as well as quality characteristics such as performance, security, reliability, usability, or other applicable characteristics.

For example, an admission system may require that applicants can submit applications electronically and that authorised staff can verify submitted documents. A performance requirement may specify an acceptable response time under defined operating conditions.

SGOU

4.4.1.8 System Architecture Definition Process

Architecture definition establishes the overall structure of the system and the relationships among its major elements.

For example, an online admission system may contain a user interface, application processing component, document management component, payment component, database, and reporting component. Architecture definition determines how these elements are organised and interact.

4.4.1.9 Design Definition Process

Design definition develops the detailed solution needed to implement the architecture and requirements. It provides a more detailed description of system elements, interfaces, data, and other design characteristics.

For example, the design of an admission application module may specify the fields required in an application form, validation rules, database structures, and interactions with other components.

4.4.1.10 System Analysis Process

System analysis supports technical decision-making by examining system characteristics, alternatives, relationships, risks, and other relevant factors.

For example, a project team may compare alternative database designs to determine which one better satisfies expected transaction volume, data integrity requirements, and maintainability needs.

4.4.1.11 Implementation Process

The implementation process transforms the defined design into software elements or other implementation results. Depending on the project, implementation may involve programming, configuration, integration of existing software, or other suitable techniques.

For example, developers may implement the admission system's applicant registration module based on the approved requirements and design specifications.

Implementation should not be considered complete merely because source code has been written. The resulting software needs to proceed through appropriate integration, verification, validation, and other relevant life cycle processes.

4.4.1.12 Integration Process

Integration brings software elements together to form larger integrated elements or a complete system.

For example, an admission system may have independently developed modules for registration, document verification, payment, and reporting. Integration establishes the required interactions among these modules and ensures that they operate together as intended.

SGOU

Integration is particularly important because individual components may work correctly in isolation but encounter problems when they interact with other components.

4.4.1.13 Verification Process

- **Meaning of Verification**

Verification is a major technical process identified in ISO/IEC/IEEE 12207:2026. It is listed as the technical process immediately following integration and before transition.

Verification focuses on obtaining objective evidence that specified requirements have been fulfilled by the relevant work products or system elements.

The central idea can be expressed as:

Verification asks whether the specified requirements have been correctly fulfilled by the relevant product or work result.

Verification is therefore closely associated with requirements, design, implementation, integration, and other technical work products.

- **Importance of Verification**

Verification helps identify defects, omissions, inconsistencies, and deviations before they propagate into later stages of the life cycle. It provides evidence that the developed result conforms to its specified requirements.

Consider a university admission system for which a requirement specifies that an authorised administrator must be able to generate an admission report containing applicant details, programme information, and admission status. During verification, the development team can examine the implemented functionality and associated test evidence to determine whether this specified requirement has been fulfilled.

Verification can be performed at different levels of the system. A software component may be verified against its requirements, an integrated set of components may be verified against its interface and functional requirements, and the complete system may be verified against applicable system requirements.

- **Verification Throughout the Life Cycle**

Verification should not be treated as an activity that occurs only after all development has been completed. Because ISO/IEC/IEEE 12207 provides processes that can be applied concurrently, iteratively, and recursively, verification can be incorporated throughout the life cycle.

For example, a design can be examined against requirements before implementation begins. An implemented component can subsequently be verified against its design and implementation requirements. After integration, the integrated product can be verified against applicable higher-level requirements.

SGOU

This approach helps identify problems earlier and reduces the likelihood that defects will remain undetected until late stages.

4.4.1.14 Validation Process

- **Meaning of Validation**

Validation is another distinct technical process in ISO/IEC/IEEE 12207:2026. It follows the transition in the technical-process sequence and is listed before operation and maintenance.

Validation focuses on determining whether the resulting system is suitable for its intended use and whether it satisfies the needs and expectations for which it was developed.

The central idea can be expressed as:

Validation asks whether the right system has been produced for its intended purpose and operational context.

Thus, verification and validation have related but different purposes.

- **Importance of Validation**

Validation provides confidence that the system is appropriate for its intended use rather than merely conforming to technical specifications.

For example, suppose an online admission system correctly implements the specified application-processing functions. Validation may involve evaluating whether actual admission staff and applicants can use the system effectively in an operationally representative environment and whether the system supports the university's actual admission procedures.

A system may satisfy individual technical requirements but still fail to support its intended operational purpose. Validation helps identify such situations.

4.4.1.15 Verification and Validation: Relationship

Verification and validation should be treated as complementary processes.

Verification primarily examines conformity with specified requirements and other applicable criteria.

Validation primarily examines suitability for the intended use and stakeholder needs.

For example, consider a university examination management system. Verification may establish that the system correctly implements the specified examination scheduling rules. Validation may determine whether examination administrators can actually use the completed system effectively to prepare, modify, publish, and manage examination schedules in the intended operational environment.

SGOU

Therefore, successful verification does not by itself establish successful validation, and successful validation does not eliminate the need for verification.

4.4.1.16 Verification and Validation in the Software Life Cycle

The relationship between the two processes can be understood through the following simplified flow:

Stakeholder Needs
↓
Requirements Definition
↓
Architecture and Design
↓
Implementation
↓
Integration
↓
Verification
↓
Transition
↓
Validation
↓
Operation and Maintenance

This representation should not be interpreted as a mandatory sequential SDLC model. ISO/IEC/IEEE 12207 explicitly allows life cycle processes to be applied concurrently, iteratively, recursively, and incrementally.

4.4.1.17 Transition Process

The transition process prepares and moves the system or relevant system elements into their intended operational environment.

For example, after an admission system has been developed and appropriately evaluated, it may need to be installed on production infrastructure, configured with operational data, and made available to authorised users.

Transition therefore connects development activities with actual operational use.

4.4.1.18 Operation Process

The operation process concerns the use of the software system in its intended operational environment.

Once a university admission system becomes operational, admission staff and applicants may use it for actual admission activities. Operational activities may also generate information about system performance, problems, user needs, and opportunities for improvement.

SGOU

4.4.1.19 Maintenance Process

Software systems often require modification after deployment. Maintenance addresses changes needed to correct problems, adapt the system to changed circumstances, or improve its capabilities.

For example, an admission system may need modification when a university changes its admission rules or introduces a new payment mechanism. Maintenance activities should preserve the integrity and intended quality of the system while implementing approved changes.

4.4.1.20 Disposal Process

The disposal process addresses the retirement of a software system or relevant system elements when they are no longer required.

Disposal should consider issues such as the termination of operational use, handling of relevant data and information, removal or decommissioning of system elements, and other applicable organisational requirements.

For example, when an old admission platform is replaced by a new system, the organisation needs to plan the retirement of the old system rather than simply stopping its use without considering retained data and operational dependencies.

4.4.1.21 Tailoring of Processes

ISO/IEC/IEEE 12207 provides a comprehensive set of processes from which organisations can select an appropriate subset for their particular objectives.

Tailoring is important because software projects differ considerably in size, complexity, criticality, technology, organisational environment, and development approach.

For example, a small internal application may not require the same depth of process implementation as a safety-critical or highly regulated software system. Tailoring allows the organisation to establish an appropriate set of processes while maintaining a structured and justifiable approach.

4.4.1.22 Application with Agile Development

ISO/IEC/IEEE 12207:2026 does not require a particular software development methodology. The standard can be applied with different engineering approaches, including Agile approaches.

Therefore, organisations do not have to choose between using the standard and using Agile development. The processes can be mapped and applied within an iterative development environment.

For example, in an Agile project, requirements may evolve through successive iterations. Verification and validation activities can also be incorporated into each appropriate iteration rather than being postponed until the end of the entire development effort.

SGOU

4.4.1.23 Process Assessment and Improvement

ISO/IEC/IEEE 12207 can also support the assessment and improvement of software life cycle processes. Organisations can establish a process environment and assess whether their projects conform to the declared environment. Process assessors can also use the standard as a process reference for process assessment and improvement.

This makes the standard useful not only for individual software projects but also for organisations seeking to improve the way software is developed and managed.

4.4.1.24 Relationship with Other Standards

ISO/IEC/IEEE 12207 is related to ISO/IEC/IEEE 15288, which addresses system engineering processes. The two standards have corresponding process purposes and outcomes in relevant areas, while their activities and tasks differ according to whether the focus is software engineering or systems engineering.

The standard also states that its requirements are intended to be compatible with quality management, service management, IT asset management, and information security management standards such as ISO 9001, ISO/IEC 20000-1, ISO/IEC 19770-1, and ISO/IEC 27001.

4.4.1.25 Important Characteristics of ISO/IEC/IEEE 12207

ISO/IEC/IEEE 12207 should be understood as a process framework rather than a fixed SDLC model. It does not prescribe Waterfall, V-Model, Agile, Spiral, or any other specific methodology.

Its processes can be selected and tailored according to project needs. They can also operate concurrently and iteratively. This flexibility allows the framework to be used across different types of software projects and organisational environments.

The standard also emphasises the involvement of stakeholders throughout the life cycle. Its overall purpose is to support the development and management of software systems that satisfy relevant stakeholder needs and provide appropriate quality.

ISO/IEC/IEEE 12207:2026 establishes a common framework for software life cycle processes covering the complete life cycle of software systems, products, and services. It can be applied to acquisition, development, operation, maintenance, support, and retirement.

The standard organises software life cycle activities into agreement, organisational project-enabling, technical management, and technical processes. Among the technical processes, verification and validation are particularly important for software quality. Verification provides evidence that specified requirements have been fulfilled, while validation evaluates whether the resulting system is suitable for its intended purpose and operational context.

A major strength of ISO/IEC/IEEE 12207 is that it does not impose a single development methodology. Instead, organisations can select and tailor appropriate processes and apply them within different life cycle approaches, including Agile and iterative development.

SGOU

Overall, the standard provides a structured foundation for managing software throughout its life cycle while allowing organisations to adapt the processes to their particular projects, products, services, and organisational objectives.

4.4.2 Capability Maturity Model Integration (CMMI)

The Capability Maturity Model Integration (CMMI) is a globally recognised process improvement framework that enables organisations to develop, assess, and continuously enhance their business and software development processes. It integrates best practices from multiple disciplines, including software engineering, systems engineering, product development, and project management, into a single, unified framework. By implementing CMMI, organisations can improve product quality, increase customer satisfaction, minimise project risks, and achieve greater operational efficiency.

CMMI was developed as an enhanced version of the original Capability Maturity Model (CMM). While CMM primarily focused on software development, CMMI integrates several specialised maturity models into a comprehensive framework that can be applied across different organisational functions. This integrated approach allows organisations to improve multiple business processes simultaneously while ensuring consistency and standardisation.

4.4.2.1 What is Capability Maturity Model Integration (CMMI)?

Capability Maturity Model Integration (CMMI) is a process improvement model that helps organisations establish efficient, predictable, and continuously improving processes. It provides a structured methodology for evaluating the maturity of organisational processes and offers guidance for systematic process enhancement.

Unlike the original CMM, which addressed individual disciplines separately, CMMI combines best practices from various domains such as Software Engineering CMM, Systems Engineering CMM, Integrated Product Development, and People CMM into a single framework. This integration eliminates duplication, simplifies process management, and enables organisations to adopt a unified improvement strategy that aligns with their business objectives.

Organisations implementing CMMI can identify weaknesses in their existing processes, establish standardised procedures, monitor process performance, and continuously optimise their operations to deliver high-quality products and services.

4.4.2.2 Objectives of CMMI

The primary objectives of implementing CMMI include:

- Satisfying customer requirements by delivering reliable and high-quality products.
- Increasing customer confidence through consistent process performance.
- Creating long-term value for investors and stakeholders.

SGOU

- Enhancing organisational productivity and operational efficiency.
- Improving the quality of products and services through standardised processes.
- Supporting continuous process improvement across all organisational activities.
- Strengthening the organisation's competitive position and market reputation.
- Reducing development risks, project delays, and operational costs.

4.4.2.3 CMMI Representations

CMMI provides two different approaches for process improvement, known as **Staged Representation** and **Continuous Representation**. Organisations can choose either approach depending on their business goals and improvement strategies.See fig. 4.4.1

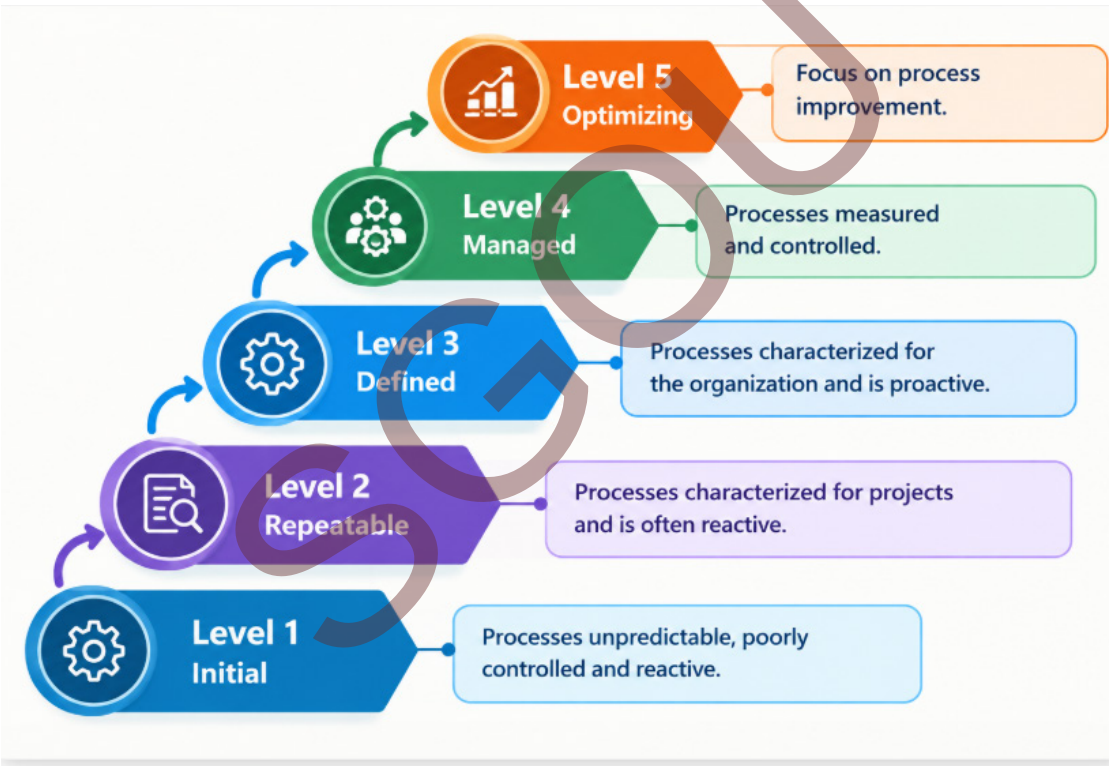


Fig. 4.4.1 CMMI Framework

1. Staged Representation

The staged representation provides a predefined sequence of process improvements based on maturity levels. Each maturity level serves as the foundation for progressing to the next level, allowing organisations to improve their processes in a structured and systematic manner.

Characteristics

- Uses a predefined collection of process areas.
- Provides a clear roadmap for organisational process improvement.
- Organises improvements into five maturity levels.
- Helps organisations achieve gradual and controlled process enhancement.
- Enables comparison of process maturity among different organisations.
- Suitable for organisations seeking organisation-wide process improvement.

2. Continuous Representation

The continuous representation focuses on improving individual process areas instead of following a fixed sequence. Organisations can prioritise process areas according to their specific business needs and risks.

Characteristics

- Allows organisations to select individual process areas for improvement.
- Measures improvement using capability levels rather than maturity levels.
- Supports flexible implementation strategies.
- Enables comparison of organisations based on individual process capabilities.
- Helps prioritise critical processes that require immediate improvement.
- Supports risk-based process improvement.

4.4.2.4 CMMI Maturity Levels

In the staged representation, organisational process maturity is measured using five maturity levels.

Maturity Level 1 – Initial

At this level, organisational processes are unpredictable, poorly documented, and inconsistently executed. Projects often depend on individual efforts rather than established procedures, resulting in frequent delays, cost overruns, and quality issues.

Characteristics

- Processes are informal and poorly managed.
- Activities are reactive rather than planned.
- No standardised procedures exist.
- High project risk and uncertainty.
- Product quality is inconsistent.

SGOU

Maturity Level 2 – Managed

Organisations establish basic project management practices to plan, monitor, and control project activities. Requirements are documented, and project execution follows approved plans.

Characteristics

- Requirements are managed systematically.
- Project planning and monitoring are performed.
- Work products are documented.
- Basic quality assurance practices are introduced.
- Risks are reduced compared to Level 1.

Maturity Level 3 – Defined

At this stage, organisational processes are standardised and documented across all projects. Standard operating procedures, development guidelines, and organisational policies are consistently followed.

Characteristics

- Processes are well documented and standardised.
- Organisational standards are established.
- Employees follow common procedures.
- Knowledge sharing is encouraged.
- Product quality becomes more consistent.

Maturity Level 4 – Quantitatively Managed

Organisations begin measuring process performance using statistical and quantitative techniques. Process behaviour is analysed to improve predictability and quality.

Characteristics

- Process performance is measured using metrics.
- Statistical methods are used for quality control.
- Quantitative objectives are established.
- Performance variations are identified and controlled.
- Decision-making becomes data-driven.

Maturity Level 5 – Optimising

This represents the highest level of organisational maturity. Continuous process improvement becomes part of the organisational culture through innovation, performance analysis, and proactive problem-solving.

SGOU

Characteristics

- Continuous process improvement is emphasised.
- Innovative technologies and practices are adopted.
- Root causes of problems are systematically eliminated.
- Organisational learning is promoted.
- Highest product quality and lowest operational risk are achieved.

4.4.2.5 CMMI Capability Levels

For organisations adopting the continuous representation, CMMI defines six capability levels that measure the maturity of individual process areas.

Capability Level 0 – Incomplete

The process is either partially implemented or not implemented at all. Many process objectives remain unachieved.

Characteristics

- Process execution is incomplete.
- Specific goals are not fully satisfied.
- No standardised practices exist.

Capability Level 1 – Performed

The process is implemented and performs its intended purpose, although consistency and predictability may still be limited.

Characteristics

- Process activities are executed.
- Basic objectives are achieved.
- Performance may vary across projects.
- Serves as the starting point for improvement.

Capability Level 2 – Managed

Processes are planned, monitored, and controlled to ensure that organisational objectives are consistently achieved.

Characteristics

- Activities are monitored regularly.
- Performance is measured.
- Resources are managed effectively.
- Quality, schedule, and cost objectives are tracked.

SGOU

Capability Level 3 – Defined

Processes are standardised according to organisational guidelines and are consistently implemented across projects.

Characteristics

- Standard operating procedures are followed.
- Organisational best practices are adopted.
- Process consistency is improved.

Capability Level 4 – Quantitatively Managed

Statistical methods and quantitative measurements are used to monitor and control process performance.

Characteristics

- Performance is measured using metrics.
- Statistical analysis supports decision-making.
- Quantitative quality goals are established.

Capability Level 5 – Optimising

Organisations continuously improve process performance through innovation, feedback, and organisational learning.

Characteristics

- Continuous improvement is encouraged.
- Root causes of defects are eliminated.
- Innovative solutions are implemented.
- Organisational performance is continually enhanced.

4.4.2.6 Benefits of CMMI

Implementing the Capability Maturity Model Integration (CMMI) framework can provide organisations with several significant benefits. It helps improve the quality of software and products by promoting systematic and well-defined processes. It also contributes to greater customer satisfaction by enabling organisations to deliver products and services that better meet customer expectations.

CMMI supports effective project planning and management, helping teams organise activities, monitor progress, and control project performance. By identifying and managing potential problems systematically, it can reduce project risks, failures, and unexpected outcomes. Standardised processes also improve productivity and operational efficiency by providing consistent approaches for performing organisational activities.

CMMI promotes effective utilisation of resources, including people, time, technology, and infrastructure. Clearly defined processes and responsibilities can facilitate better

SGOU

communication and coordination among project teams. Improved process management can also contribute to the timely delivery of products and services. Over time, consistent process improvement and improved delivery performance can strengthen an organisation's reputation, competitiveness, and ability to achieve its business objectives.

4.4.2.7 Applications of CMMI

CMMI is widely applied across various industries, including:

- Software development organisations.
- Information Technology (IT) companies.
- Manufacturing industries.
- Aerospace and defence organisations.
- Healthcare systems.
- Banking and financial institutions.
- Government agencies.
- Telecommunications companies.
- Educational institutions.

Capability Maturity Model Integration (CMMI) is a comprehensive process improvement framework that enables organisations to establish standardised, measurable, and continuously improving processes. By adopting either the staged or continuous representation, organisations can systematically enhance process maturity and capability while reducing risks and improving product quality. Through its emphasis on process standardisation, quantitative management, and continuous optimisation, CMMI helps organisations achieve operational excellence, customer satisfaction, and sustainable business growth. As a result, it has become one of the most widely accepted frameworks for process improvement and quality management across industries.

SGOU

Recap

- ISO/IEC/IEEE 12207:2026 provides a common framework for managing software life cycle processes.
- The standard covers software activities from conception and development to operation, maintenance, and retirement.
- ISO/IEC/IEEE 12207 does not prescribe a specific SDLC model or development methodology.
- The major process groups include agreement, organisational project-enabling, technical management, and technical processes.
- Technical processes cover activities such as requirements definition,

architecture, design, implementation, integration, verification, validation, operation, and maintenance.

- Verification provides evidence that specified requirements have been fulfilled by the relevant software work products or system elements.
- Validation determines whether the resulting system is suitable for its intended use and satisfies the relevant needs and expectations.
- Verification and validation are complementary processes, and both contribute to software quality.
- ISO/IEC/IEEE 12207 processes can be applied concurrently, iteratively, recursively, or incrementally according to project needs.
- Tailoring allows organisations to select and adapt appropriate processes according to project size, complexity, risks, and objectives.
- CMMI is a process improvement framework that helps organisations establish, assess, and continuously improve their processes.
- CMMI provides Staged and Continuous Representations for organisational process improvement.
- The Staged Representation uses five maturity levels: Initial, Managed, Defined, Quantitatively Managed, and Optimising.
- The Continuous Representation uses capability levels to measure and improve individual process areas.
- CMMI can help organisations improve quality, customer satisfaction, productivity, risk management, resource utilisation, and process consistency.

SGOU

Objective Type Questions

1. What standard provides a common framework for software life cycle processes?
2. What is the primary purpose of ISO/IEC/IEEE 12207?
3. Which process establishes the relationship between acquirers and suppliers?
4. Which process group provides organisational capabilities to support software projects?
5. Which technical process establishes the overall structure of a system?

6. Which process transforms a defined design into software elements?
7. Which process brings software elements together into an integrated system?
8. Which process obtains objective evidence that specified requirements have been fulfilled?
9. Which process determines whether a system is suitable for its intended use?
10. Which process prepares and moves a system into its operational environment?
11. Which process addresses changes made to software after deployment?
12. Which process deals with the retirement of a software system?
13. What approach allows organisations to select and adapt processes according to their needs?
14. Which CMMI representation uses predefined maturity levels?
15. Which CMMI maturity level focuses on continuous process improvement?

Answers to Objective Type Questions

1. ISO/IEC/IEEE 12207.
2. Framework.
3. Agreement.
4. Organisational.
5. Architecture.
6. Implementation.
7. Integration.
8. Verification.
9. Validation.
10. Transition.
11. Maintenance.
12. Disposal.
13. Tailoring.
14. Staged.
15. Optimising.

SGOU

Assignments

1. Explain the purpose and scope of ISO/IEC/IEEE 12207:2026. Describe how the standard provides a common framework for managing software throughout its life cycle.

2. Explain the major process groups in ISO/IEC/IEEE 12207:2026. Describe the role of agreement processes, organisational project-enabling processes, technical management processes, and technical processes.

3. Describe the major technical processes identified in ISO/IEC/IEEE 12207:2026, from business or mission analysis through disposal, with suitable examples.

4. Explain the Verification and Validation processes in ISO/IEC/IEEE 12207:2026. Clearly distinguish between verification and validation and discuss their importance in the software life cycle.

5. Explain how ISO/IEC/IEEE 12207:2026 can be applied to different software development approaches, including Agile and iterative development. Discuss the importance of process tailoring.

6. Explain Capability Maturity Model Integration (CMMI) and its main objectives. Discuss how CMMI helps organisations improve software quality, project management, productivity, and customer satisfaction.

7. Describe the Staged and Continuous Representations of CMMI. Explain the differences between maturity levels and capability levels with suitable examples.

8. Explain the five maturity levels of CMMI in detail. Describe the characteristics of each level from Initial to Optimising and explain how an organisation can progressively improve its processes.

SGOU

Suggested Reading

1. ISO/IEC/IEEE 12207:2026 — Systems and Software Engineering — Software Life Cycle Processes.

2. Sethi, R. (2023). Software engineering: Basic principles and best practices. Cambridge University Press.

3. O'Regan, G. (2022). Concise guide to software engineering: From fundamentals to application methods. Springer.

4. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). Essentials of software testing. Cambridge University Press.

5. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

6. Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Reference

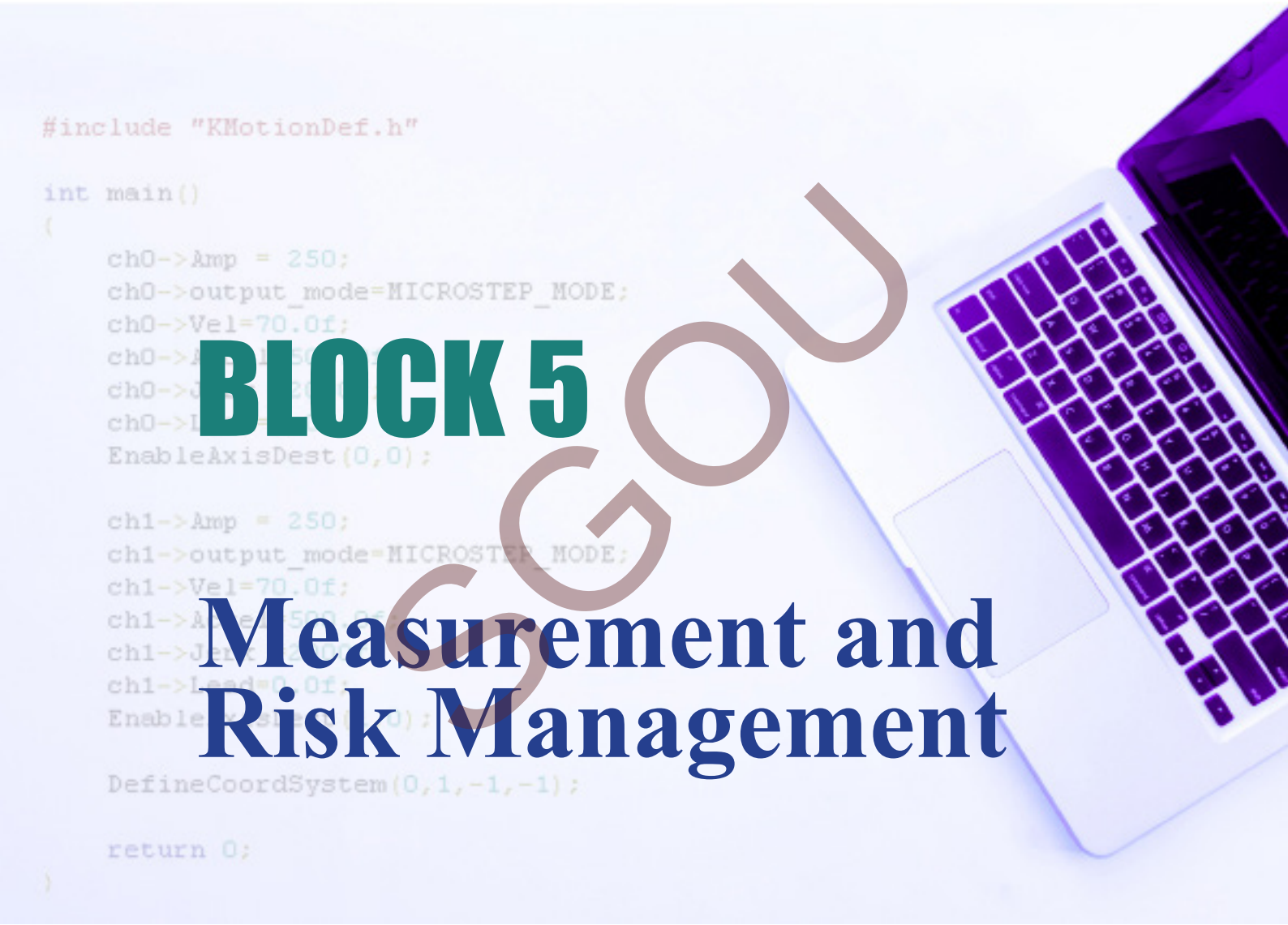
1. Jalote, P. (2025). A concise introduction to software engineering: With open source and GenAI (2nd ed.). Springer.

2. Wang, et al. (2024). Enhancing human-machine pair inspection with risk number and code inspection diagram. Software Quality Journal, 32, 939–959

3. Pressman, R. S., & Maxim, B. R. (2023). Software engineering: A practitioner's approach (9th ed.). McGraw Hill.

4. Pfleeger, S. L., & Atlee, J. M. (2020). Software engineering: Theory and practice (5th ed.). Pearson.

SGOU



BLOCK 5

Measurement and Risk Management

SGOU

UNIT 1

Foundations of Software Measurement

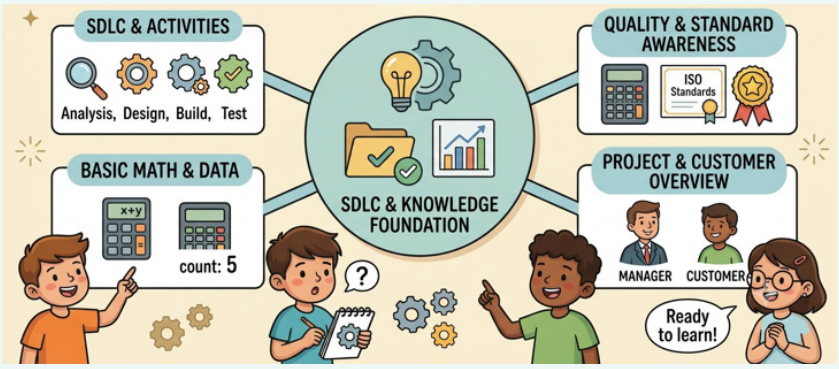
Learning Outcomes

After completion of this unit, learner will be able to:

- define software measurement and explain its importance in software development
- explain the basic concepts and types of software measurement
- describe the software measurement process in ISO/IEC/IEEE 12207 and ISO 9001
- apply software measurement techniques to evaluate software quality and project performance
- compare ISO/IEC/IEEE 12207 and ISO 9001 based on their measurement approaches

Prerequisite

A basic understanding of the Software Development Life Cycle (SDLC), software quality, and project management concepts is helpful before studying this unit. Familiarity with software development activities such as requirements analysis, design, coding, testing, and maintenance provides a strong foundation for understanding software measurement. Basic knowledge of data collection, simple calculations, and performance evaluation is also useful. For example, counting the number of software defects or measuring the time required to complete a development task provides valuable information about software quality and project performance.



SGOU

A general awareness of quality management and the role of standards in software engineering is also beneficial. Understanding concepts such as customer requirements, process improvement, and quality assurance helps in understanding the software measurement practices defined in ISO/IEC/IEEE 12207 and ISO 9001. For example, measurement data showing fewer defects and timely project completion can be used to demonstrate that a software project meets quality objectives and customer expectations.

Keywords

Software Measurement, Software Metrics, ISO/IEC/IEEE 12207, ISO 9001, Measurement Process

Discussion

5.1.1 Introduction to Software Measurement

Software engineering involves the systematic development, maintenance, and management of software systems. Throughout the Software Development Life Cycle (SDLC), organizations make critical decisions related to project planning, cost estimation, resource allocation, quality assurance, and risk management. To make these decisions effectively, organizations rely on objective evidence and measurable data rather than assumptions or subjective judgments. This approach is achieved through software measurement.

Software measurement is the systematic process of collecting, analyzing, and interpreting quantitative data related to software products, processes, and projects. It provides valuable information about the current state of a software project, helps monitor progress, supports prediction of future outcomes, and enables continuous improvement in software quality and development practices.

Software projects generate various measurable attributes, such as the number of defects, lines of code developed, test coverage, development effort, execution time, customer satisfaction, and project completion duration. These measurements help organizations evaluate software performance, verify compliance with quality standards, and determine whether project goals are being achieved.

Software measurement plays a vital role in software engineering by supporting:

- Software quality assurance through continuous evaluation of software quality and reliability.
- Project planning and scheduling by providing accurate data for resource and time management.
- Cost and effort estimation by predicting development requirements.

SGOU

- Productivity improvement through analysis of development efficiency.
- Risk management by identifying potential issues and uncertainties.
- Process improvement by evaluating and optimizing software development practices.
- Compliance with international standards such as ISO/IEC/IEEE 12207 and ISO 9001.

In today's competitive software industry, measurement-based decision-making helps organizations improve software quality, reduce development costs, optimize processes, deliver projects on time, and achieve customer satisfaction.

Definition of Software Measurement

Software measurement is the process of quantifying the characteristics of software products, development processes, and projects by assigning numerical values to specific attributes according to predefined measurement rules. It provides objective data that enables software professionals to assess quality, estimate effort, track progress, and evaluate performance throughout the software development life cycle.

By using measurable indicators instead of subjective judgments, software measurement supports informed decision-making and continuous improvement. It enables organizations to compare actual results with expected goals, identify improvement areas, and ensure that software development activities meet quality, cost, schedule, and customer requirements effectively.

5.1.1.1 Characteristics of Effective Software Measurement

Effective software measurement provides accurate and meaningful information that supports software development, quality assurance, and project management. A good measurement system should produce reliable data that helps organizations monitor performance, make informed decisions, and continuously improve software processes. Some key characteristics are:

1. **Validity:** An effective measurement should accurately measure the attribute it is intended to evaluate. The collected data must truly represent the software product, process, or project.

2. **Reliability:** The measurement should produce consistent results when repeated under the same conditions. Reliable measurements ensure that decisions are based on dependable information.

3. **Accuracy:** Software measurements should be precise and free from errors. Accurate data helps organizations evaluate performance correctly and avoid incorrect conclusions.

4. **Simplicity:** A good measurement method should be easy to understand, implement, and interpret. Simple measurement techniques reduce complexity and improve usability.

SGOU

5. **Relevance:** The measurement should focus only on information that supports project objectives and organizational goals. Relevant metrics help managers make meaningful decisions.

6. **Timeliness:** Measurement data should be collected and reported at the right time so that corrective actions can be taken quickly. Timely information supports effective project monitoring and control.

Effective software measurement is characterized by validity, reliability, accuracy, simplicity, relevance, and timeliness. These characteristics ensure that measurement data is trustworthy, useful, and capable of supporting better decision-making, quality improvement, and successful software project management.

5.1.2 The Importance of Measurement

Measurement is a fundamental activity in software engineering that provides objective and quantitative information about software products, processes, and projects. It helps organizations monitor progress, evaluate quality, improve efficiency, and make informed decisions throughout the software development life cycle. Effective measurement enables organizations to deliver reliable software while controlling cost, time, and resources. Some key points are:

1. **Improves Software Quality:** Measurement helps assess software quality by tracking defects, reliability, maintainability, and performance. Regular measurement enables developers to identify quality issues early and take corrective acions before software is released.

2. **Supports Accurate Project Planning:** Measurement provides historical and current data that helps estimate project size, development effort, cost, resources, and completion time. Accurate planning reduces the chances of project delays and budget overruns.

3. **Monitors Project Progress:** Measurement allows project managers to compare actual progress with planned milestones. Tracking schedule, budget, and task completion helps ensure that the project remains on schedule and meets its objectives.

4. **Enhances Decision-Making:** Reliable measurement data supports informed decision-making at both technical and managerial levels. Managers use measurement results to allocate resources, prioritize tasks, manage risks, and improve project performance.

5. **Increases Productivity:** Measurement evaluates the efficiency of developers and development teams by analyzing work completed, defects fixed, and development speed. This information helps organizations identify improvement opportunities and increase productivity.

6. **Reduces Project Risks:** Continuous measurement helps identify potential risks such as schedule delays, cost overruns, increasing defects, or declining

SGOU

productivity. Early detection enables timely corrective actions and minimizes the impact of project risks.

7. **Supports Continuous Process Improvement:** Measurement provides valuable feedback on the effectiveness of software development processes. Organizations analyze measurement results to improve development practices, optimize workflows, and enhance overall software quality.

8. **Increases Customer Satisfaction:** Measurement ensures that software meets customer requirements by evaluating quality, performance, reliability, and delivery schedules. Delivering high-quality software on time improves customer confidence and satisfaction.

Measurement is essential for successful software development because it provides reliable information for evaluating quality, managing projects, improving productivity, and reducing risks. By supporting continuous improvement and evidence-based decision-making, software measurement helps organizations deliver high-quality software that meets both business objectives and customer expectations.

5.1.3 Fundamental Concepts of Software Measurement

Software measurement is based on a set of fundamental concepts that help organizations collect, analyze, and interpret data about software products, processes, and projects. The three key concepts are **measures, metrics, and indicators**. Although these terms are closely related, they have different meanings and purposes. Understanding these concepts helps in selecting appropriate measurement techniques and making informed decisions during software development. Fig 5.1.1 depicted as the three key concepts in software measurement.

1. Measures

A **measure** is a direct numerical value obtained by measuring a specific attribute of a software product, process, or project. It is the basic unit of software measurement and is collected without any calculations.

Characteristics

- Represents raw numerical data.
- Directly obtained from observation or counting.
- Used as input for calculating software metrics.
- Simple and objective.

Examples

- Number of lines of code (LOC)
- Number of defects found
- Number of test cases executed
- Development time (hours)

SGOU

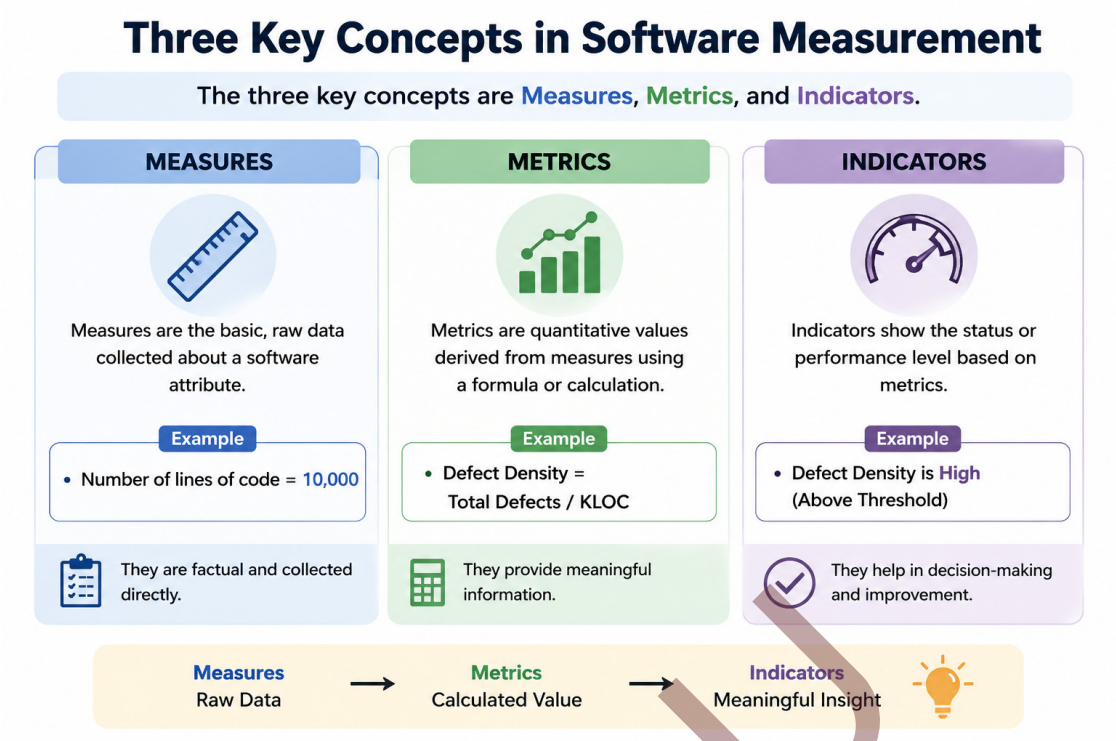


Fig 5.1.1 Three key concepts in Software Measurement

2. Metrics

A **metric** is a quantitative value calculated by combining one or more measures. Metrics provide meaningful information that helps evaluate software quality, productivity, performance, or process effectiveness.

Characteristics

- Derived from one or more measures.
- Helps compare performance.
- Supports decision-making.
- Used to monitor software quality and project progress.

Examples

- Defect Density = Number of Defects ÷ Lines of Code
- Productivity = Lines of Code ÷ Development Effort
- Test Coverage = (Executed Test Cases ÷ Total Test Cases) × 100
- Defect Removal Efficiency (DRE)

3. Indicators

An **indicator** is a higher-level interpretation of one or more metrics that provides insight into the overall status, trends, or performance of a software project. Indicators help managers identify problems, monitor progress, and make strategic decisions.

SGOU

Characteristics

- Derived from metrics.
- Shows trends and performance.
- Supports project monitoring.
- Helps identify risks and improvement areas.

Examples

- Project status (On Schedule / Behind Schedule)
- Customer satisfaction level
- Overall project health
- Risk level (Low, Medium, High)

Measures, metrics, and indicators are fundamental concepts in software measurement that help evaluate the performance, quality, and progress of software products, processes, and projects. Although these terms are closely related, each has a distinct purpose and provides different levels of information for effective decision-making. Table 5.1.1 shows the major differences.

Table 5.1.1 Difference Between Measures, Metrics, and Indicators

Feature	Measure	Metric	Indicator
Meaning	Raw numerical value	Calculated value using measures	Interpretation of metrics
Purpose	Collect basic data	Evaluate performance	Support decision-making
Data Type	Direct measurement	Derived measurement	Analytical information
Complexity	Simple	Moderate	High
Example	200 defects	Defect Density = 2 defects/KLOC	Software quality is improving

Relationship among Measures, Metrics, and Indicators

The three concepts are closely connected: **Measures → Metrics → Indicators**

Let's consider,

- Measures provide the raw data.
- Metrics analyze the raw data to evaluate performance.
- Indicators interpret the metrics to support management decisions.

Example,

- **Measure:** 25 defects found in 10,000 lines of code.
- **Metric:** Defect Density = 25 ÷ 10 = 2.5 defects/KLOC.
- **Indicator:** The defect density is low, indicating good software quality.

SGOU

Measures, metrics, and indicators are the foundation of software measurement. Measures provide raw data, metrics transform that data into meaningful values, and indicators interpret the results to guide decision-making. Together, they help software organizations monitor project performance, improve quality, manage risks, and achieve successful software development outcomes.

5.1.3.1 Types of Software Metrics

Software metrics are quantitative measures used to evaluate different aspects of software development and management. They help organizations assess the quality of software products, the efficiency of development processes, and the progress of software projects. Software metrics are broadly classified into **Product Metrics**, **Process Metrics**, and **Project Metrics** (Fig 5.1.2).

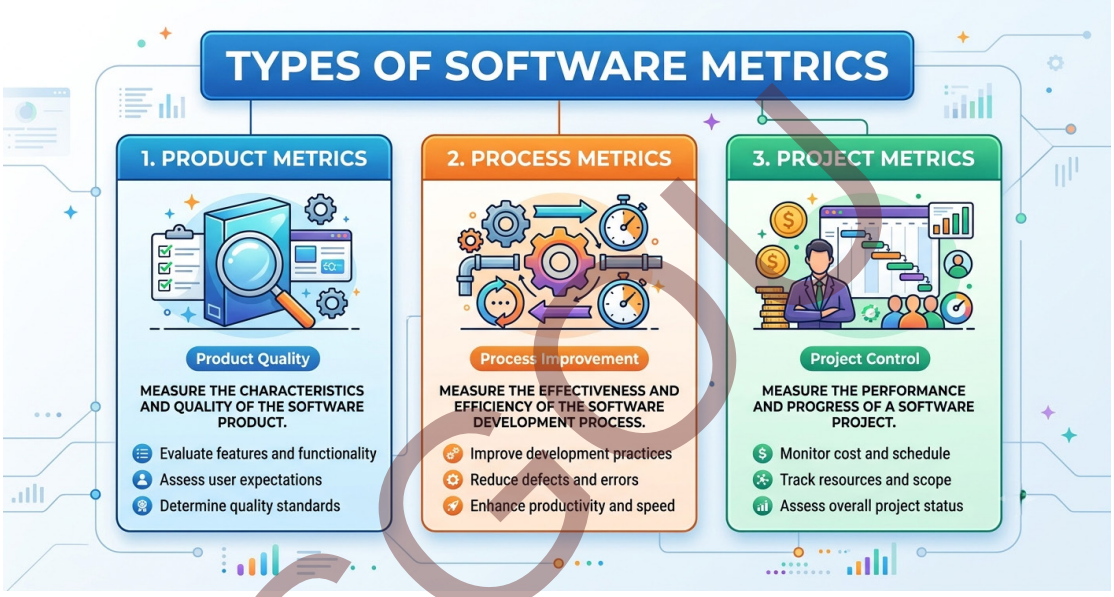


Fig 5.1.2 Types of Software metrics

1. Product Metrics

Product metrics measure the characteristics and quality of the software product. These metrics help evaluate whether the software meets the required quality standards and user expectations.

Examples

- Lines of Code (LOC)
- Function Points (FP)
- Defect Density
- Response Time
- Reliability

SGOU

2. Process Metrics

Process metrics measure the effectiveness and efficiency of the software development process. They help organizations improve development practices, reduce errors, and enhance productivity.

Examples

- Defect Removal Efficiency (DRE)
- Testing Efficiency
- Code Review Effectiveness
- Build Success Rate
- Process Compliance Rate

3. Project Metrics

Project metrics measure the performance and progress of a software project. These metrics help project managers monitor cost, schedule, resources, and overall project status.

Examples

- Project Cost
- Development Effort (Person-hours)
- Budget Variance
- Team Productivity
- Resource Utilization

The cmparison of Product Metrics, Process Metrics, and Project Metrics is shown in Table 5.1.2.

Table 5.1.2 Comparative overview of Product, Process, and Project Metrics

Feature	Product Metrics	Process Metrics	Project Metrics
Focus	Software product	Development process	Software project
Purpose	Measure software quality	Improve development process	Monitor project performance
Used By	Developers, Testers	Process Managers, QA Teams	Project Managers
Measures	Quality, size, complexity	Process efficiency, testing, reviews	Cost, schedule, effort, resources
Examples	LOC, Defect Density, Reliability	DRE, Testing Efficiency, Build Success Rate	Cost, Schedule Variance, Productivity

SGOU

5.1.3.2 Direct and Indirect Measurements

Software measurement can be classified into **direct measurements** and **indirect measurements** based on how the values are obtained. Direct measurements are collected by observing or counting specific software attributes, whereas indirect measurements are derived by combining one or more direct measurements. Both types of measurements are important for evaluating software quality, project performance, and process efficiency (Fig 5.1.3).

1. Direct Measurements

Direct measurements are values obtained by directly measuring or counting a software attribute without performing any calculations. These measurements provide raw data that forms the basis for software metrics.

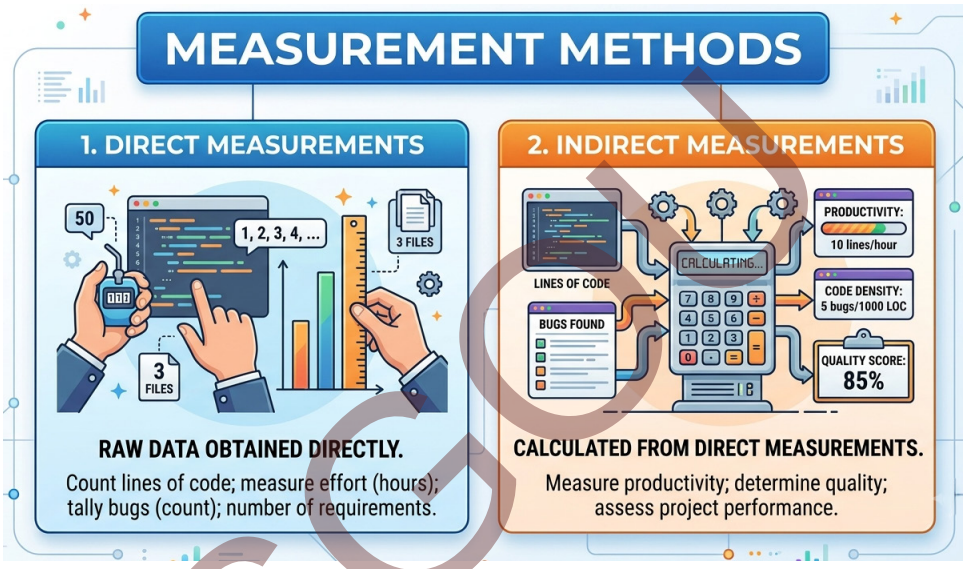


Fig 5.1.3 Direct and Indirect Measurements

2. Indirect Measurements

Indirect measurements are values obtained by combining or calculating one or more direct measurements. They provide meaningful information about software quality, productivity, and project performance.

Software measurement uses both direct and indirect measurements to assess software quality and project performance. Table 5.1.3 highlights the key differences between Direct and Indirect Measurements.

Table 5.1.3 Difference Between Direct and Indirect Measurements

Feature	Direct Measurements	Indirect Measurements
Definition	Obtained by direct observation or counting	Calculated from one or more direct measurements
Nature	Raw data	Derived information

Calculation	No calculation required	Requires formulas or computations
Purpose	Collect basic measurement data	Evaluate quality, productivity, and performance
Examples	LOC, Number of Defects, Development Time	Defect Density, Productivity, Test Coverage

5.1.4 Software Measurement According to ISO/IEC/ IEEE 12207

Software measurement according to ISO/IEC/IEEE 12207 focuses on the systematic collection, analysis, and use of measurement data to evaluate software products, processes, and projects throughout the software life cycle. The standard provides a structured framework that enables organizations to monitor performance, improve software quality, support informed decision-making, and achieve continuous process improvement through effective measurement practices.

5.1.4.1 Overview of ISO/IEC/IEEE 12207

ISO/IEC/IEEE 12207 is an internationally recognized standard that provides a comprehensive framework for managing the software life cycle processes. It was jointly developed by the International Organization for Standardization (ISO), the International Electrotechnical Commission (IEC), and the Institute of Electrical and Electronics Engineers (IEEE). The standard defines the processes, activities, and tasks required for software acquisition, development, operation, maintenance, and retirement. It serves as a common guideline for software developers, project managers, customers, and organizations to ensure consistency and quality throughout the software development life cycle.

The standard also emphasizes software measurement as an important supporting process for monitoring and improving software projects. It encourages organizations to collect, analyze, and use measurement data to evaluate software quality, project performance, and process effectiveness. By following ISO/IEC/IEEE 12207, organizations can improve project management, reduce development risks, ensure compliance with international standards, and continuously enhance their software development processes, resulting in reliable and high-quality software products.

5.1.4.2 Measurement Process in ISO/IEC/IEEE 12207

The Measurement Process in ISO/IEC/IEEE 12207 provides a structured approach for collecting, analyzing, and reporting quantitative information throughout the software life cycle. It enables organizations to evaluate the performance of software products, development processes, and projects using objective data. By following a systematic measurement process, organizations can monitor progress, identify areas for improvement, support evidence-based decision-making, and ensure that software development activities align with quality and organizational objectives.

SGOU

Objectives of the Measurement Process

The measurement process in ISO/IEC/IEEE 12207 is designed to provide reliable and quantitative information that supports effective software project management and continuous improvement. Its objectives ensure that measurement activities contribute to better quality, informed decision-making, and efficient software development.

1. **Support Informed Decision-Making:** Provide accurate measurement data to help managers and stakeholders make objective decisions.

2. **Evaluate Software Quality:** Measure quality attributes such as reliability, maintainability, performance, and defect levels.

3. **Monitor Project Performance:** Track project progress, schedule, cost, effort, and resource utilization throughout the software life cycle.

4. **Assess Process Effectiveness:** Evaluate the efficiency of software development processes and identify areas for improvement.

5. **Identify Risks and Issues:** Detect potential project risks, delays, and quality problems at an early stage for timely corrective action.

6. **Support Continuous Improvement:** Use measurement results to improve software processes, increase productivity, and enhance project outcomes.

7. **Ensure Compliance with Standards:** Verify that software development activities conform to organizational policies and international standards.

8. **Improve Customer Satisfaction:** Ensure that software products meet customer requirements by monitoring quality and project performance.

5.1.4.3 Activities in the Measurement Process

The Measurement Process in ISO/IEC/IEEE 12207 consists of a series of systematic activities that help organizations collect, analyze, and use measurement data effectively. These activities ensure that measurement supports project management, quality improvement, and informed decision-making throughout the software life cycle (Fig 5.1.4).

SGOU

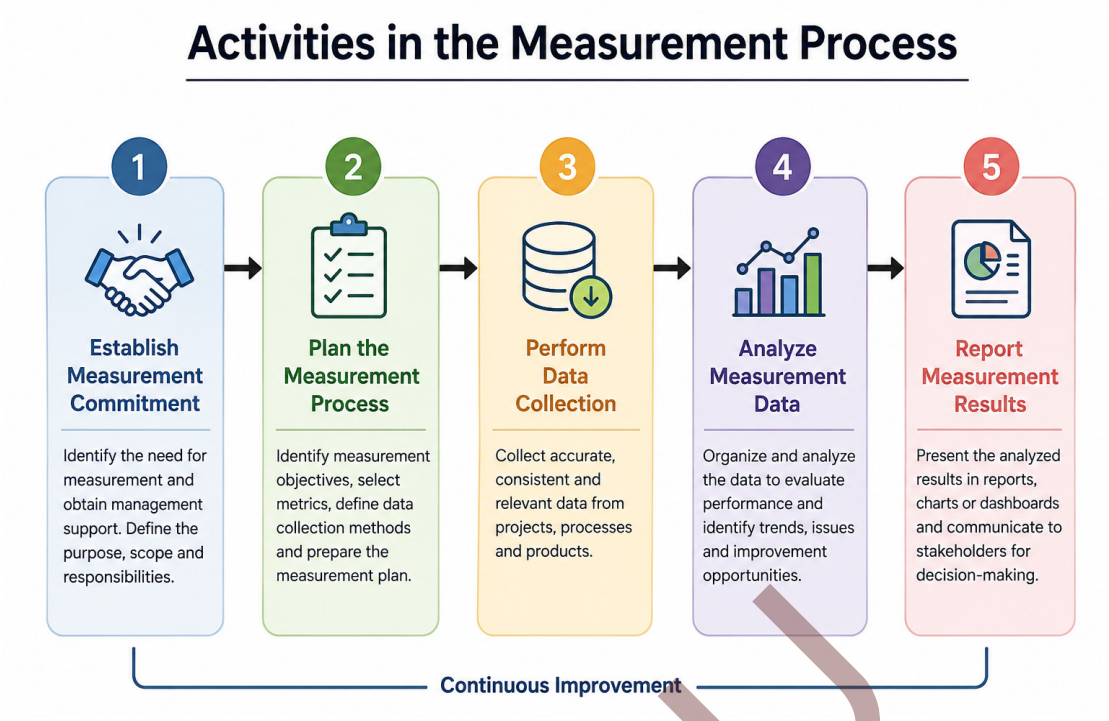


Fig 5.1.4 Activities in the Measurement Process

1. Establish Measurement Commitment

The organization identifies the need for measurement and obtains management support for measurement activities. It defines the purpose, scope, responsibilities, and resources required for implementing the measurement process.

2. Plan the Measurement Process

A measurement plan is prepared by identifying measurement objectives, selecting appropriate metrics, defining data collection methods, assigning responsibilities, and establishing reporting procedures. Proper planning ensures that relevant and reliable data is collected.

3. Perform Data Collection

Measurement data is collected from software products, development processes, and project activities. The collected data should be accurate, consistent, and relevant to the defined measurement objectives. Examples of collected data include:

- Number of defects
- Development effort
- Project cost
- Testing results
- Schedule progress

SGOU

4. Analyze Measurement Data

The collected data is organized and analyzed to evaluate software quality, process performance, and project progress. Analysis helps identify trends, strengths, weaknesses, risks, and opportunities for improvement.

5. Report Measurement Results

The analyzed results are presented to project managers, developers, and stakeholders in the form of reports, charts, dashboards, or graphs. Reporting enables informed decision-making and supports continuous improvement of software products and processes.

The measurement process in software engineering consists of a series of systematic activities that help in collecting, analyzing, and interpreting relevant data for effective decision-making and process improvement. These activities ensure that software projects are monitored and evaluated based on defined measurement objectives. Table 5.1.5 provides a summarized overview of the key activities involved in the software measurement process.

Table 5.1.5 Key activities involved in the software measurement process

Activity	Purpose
Establish Measurement Commitment	Define objectives, responsibilities, and management support.
Plan the Measurement Process	Select metrics and prepare the measurement plan.
Perform Data Collection	Gather accurate and relevant measurement data.
Analyze Measurement Data	Evaluate performance and identify improvement opportunities.
Report Measurement Results	Communicate findings to stakeholders for decision-making.

The activities of the measurement process in ISO/IEC/IEEE 12207 provide a structured approach to collecting, analyzing, and reporting software measurement data. By following these activities, organizations can monitor project performance, improve software quality, support effective decision-making, and achieve continuous process improvement throughout the software development life cycle.

5.1.4.4 Benefits of Applying ISO/IEC/IEEE 12207

Applying ISO/IEC/IEEE 12207 helps organizations establish a systematic and standardized approach to managing software life cycle processes. The standard improves software quality, enhances project management, supports effective measurement practices, and promotes continuous improvement. By following its guidelines, organizations can develop reliable software while meeting customer and industry requirements.

SGOU

1. **Improves Software Quality:** The standard defines well-structured software development processes that help produce reliable, secure, and high-quality software products.

2. **Standardizes Software Development:** It provides a common framework for software life cycle activities, ensuring consistency and uniformity across software projects.

3. **Enhances Project Management:** ISO/IEC/IEEE 12207 supports better planning, scheduling, monitoring, and control of software projects, leading to timely project completion.

4. **Supports Effective Measurement:** The standard encourages organizations to collect, analyze, and use measurement data to evaluate software products, processes, and project performance.

5. **Reduces Project Risks:** By monitoring software activities and identifying issues early, organizations can minimize defects, delays, and project failures.

6. **Promotes Continuous Improvement:** Measurement results and process evaluations help organizations identify improvement opportunities and enhance software development practices over time.

7. **Ensures Compliance with International Standards:** Following ISO/IEC/IEEE 12207 helps organizations comply with internationally recognized software engineering standards and best practices.

8. **Increases Customer Satisfaction:** The standard helps organizations deliver high-quality software that meets customer requirements, resulting in greater customer confidence and satisfaction.

Applying ISO/IEC/IEEE 12207 enables organizations to improve software quality, standardize development processes, strengthen project management, and support continuous improvement. Its structured approach to software life cycle management and measurement contributes to the successful delivery of reliable software products while ensuring compliance with international standards and meeting customer expectations.

5.1.5 Measurement According to ISO 9001

ISO 9001 is an internationally recognized standard for Quality Management Systems (QMS) developed by the International Organization for Standardization (ISO). It provides a framework for organizations to establish, implement, maintain, and continually improve their quality management practices. The standard is applicable to organizations of all sizes and industries and focuses on consistently delivering products and services that meet customer and regulatory requirements.

SGOU

ISO 9001 is based on key quality management principles such as customer focus, leadership, engagement of people, process approach, improvement, evidence-based decision-making, and relationship management. It encourages organizations to monitor and measure process performance, analyze results, and take corrective actions to improve efficiency and effectiveness. By implementing ISO 9001, organizations can enhance product quality, increase customer satisfaction, improve operational performance, and achieve continual improvement.

5.1.5.1 Measurement, Analysis, and Evaluation Requirements

ISO 9001 requires organizations to establish effective methods for **measurement, analysis, and evaluation** to ensure that the Quality Management System (QMS) achieves its intended results. These activities help organizations monitor performance, identify improvement opportunities, and make evidence-based decisions for continual improvement.

Key Requirements

1. **Measure Process Performance:** Monitor and measure processes to ensure they operate effectively and achieve planned results.
2. **Evaluate Product and Service Quality:** Verify that products and services meet customer requirements and quality standards.
3. **Monitor Customer Satisfaction:** Collect and analyze customer feedback, complaints, and survey results to assess customer satisfaction.
4. **Analyze Performance Data:** Analyze data from processes, products, and customer feedback to identify trends and support decision-making.
5. **Conduct Internal Audits:** Perform regular internal audits to verify compliance with ISO 9001 requirements and organizational policies.
6. **Review Quality Objectives:** Monitor and evaluate quality objectives to determine whether organizational goals are being achieved.
7. **Implement Corrective Actions:** Identify nonconformities and take corrective actions to eliminate their causes and prevent recurrence.
8. **Promote Continual Improvement:** Use measurement and evaluation results to improve processes, products, services, and the overall Quality Management System.

The measurement, analysis, and evaluation requirements of ISO 9001 ensure that organizations continuously monitor their performance and improve their Quality Management System. These requirements support better quality, increased customer satisfaction, and continual organizational improvement.

SGOU

5.1.5.2 Monitoring Customer Satisfaction

Monitoring customer satisfaction is a key requirement of **ISO 9001** that helps organizations determine how well they fulfill customer needs and expectations. It involves collecting, evaluating, and using customer feedback to improve products, services, and business processes. Effective monitoring strengthens customer relationships and supports continual improvement. Key points are:

1. **Gather Customer Feedback:** Collect opinions through surveys, questionnaires, interviews, and online feedback platforms.

2. **Measure Customer Expectations:** Compare customer expectations with the actual performance of products and services.

3. **Evaluate Service Quality:** Assess the quality of customer service, product delivery, and after-sales support.

4. **Monitor Customer Loyalty:** Track repeat purchases, customer retention, and long-term business relationships.

5. **Review Customer Complaints and Suggestions:** Analyze complaints and suggestions to identify opportunities for improvement and innovation.

6. **Measure Response Time:** Evaluate how quickly the organization responds to customer inquiries, requests, and complaints.

7. **Identify Improvement Opportunities:** Use customer satisfaction data to improve business processes, product quality, and service delivery.

8. **Support Business Growth:** High customer satisfaction enhances the organization's reputation, attracts new customers, and contributes to long-term business success.

Monitoring customer satisfaction provides valuable insights into customer experiences and organizational performance. By understanding customer expectations and acting on their feedback, organizations can improve quality, build customer trust, and achieve sustainable growth while meeting the requirements of ISO 9001.

5.1.5.3 Continuous Improvement Through Measurement

Continuous improvement through measurement is a fundamental principle of ISO 9001 that focuses on using measurement data to enhance the effectiveness of the Quality Management System (QMS). By regularly measuring and evaluating organizational performance, businesses can identify weaknesses, implement improvements, and achieve better quality, efficiency, and customer satisfaction. Some key points are:

1. **Measure Organizational Performance:** Collect performance data to evaluate the effectiveness of processes, products, and services.

SGOU

2. **Identify Performance Gaps:** Compare actual results with planned objectives to detect areas that require improvement.

3. **Use Data for Decision-Making:** Analyze measurement results to make informed and evidence-based management decisions.

4. **Implement Corrective and Preventive Actions:** Eliminate the causes of existing problems and prevent similar issues from occurring in the future.

5. **Optimize Business Processes:** Improve process efficiency by reducing waste, minimizing delays, and increasing productivity.

6. **Monitor Improvement Results:** Continuously evaluate the effectiveness of implemented improvements to ensure desired outcomes are achieved.

7. **Encourage Innovation and Learning:** Use measurement findings to introduce new ideas, improve practices, and enhance employee knowledge and skills.

8. **Achieve Long-Term Organizational Excellence:** Continuous measurement and improvement help organizations maintain high-quality standards, strengthen competitiveness, and achieve sustainable success.

Continuous improvement through measurement enables organizations to monitor performance, identify improvement opportunities, and enhance the effectiveness of their Quality Management System. By using reliable measurement data to guide improvement initiatives, organizations can deliver higher-quality products and services, increase customer satisfaction, and achieve long-term business success.

5.1.5.4 Documentation and Evidence of Measurement

Documentation and evidence of measurement are essential requirements of **ISO 9001** for maintaining an effective **Quality Management System (QMS)**. Organizations must record measurement activities and maintain reliable evidence to demonstrate that processes, products, and services comply with specified requirements. Proper documentation supports transparency, accountability, audits, and continual improvement. Key points are:

1. **Maintain Measurement Records:** Keep accurate records of all measurement results, inspections, tests, and monitoring activities.

2. **Document Measurement Procedures:** Prepare documented procedures that explain how measurement activities are planned, performed, and controlled.

3. **Provide Objective Evidence:** Preserve evidence such as reports, inspection records, test results, and audit findings to demonstrate compliance.

4. **Ensure Record Accuracy:** Verify that all measurement records are complete, accurate, and updated regularly.

5. **Store Records Securely:** Protect measurement documents from loss, damage,

SGOU

or unauthorized access through proper storage and control.

6. **Support Internal and External Audits:** Maintain documented evidence that can be reviewed during quality audits and certification assessments.

7. **Ensure Traceability:** Keep records that allow measurement results to be traced back to specific processes, products, equipment, or activities.

8. **Use Documentation for Improvement:** Analyze documented measurement data to identify trends, evaluate performance, and support continual improvement initiatives.

Documentation and evidence of measurement provide reliable proof that quality management activities are carried out effectively and in accordance with ISO 9001 requirements. Well-maintained records support compliance, improve traceability, facilitate audits, and provide valuable information for enhancing organizational performance and achieving continual improvement.

5.1.6 Comparison of Software Measurement Standards

Software measurement and quality standards provide structured guidelines for assessing, managing, and improving software products, processes, and projects. These standards help organizations establish systematic approaches to ensure quality, efficiency, and continuous improvement throughout the software development lifecycle. Different standards address specific aspects of software engineering, including measurement activities, process management, quality assurance, and organizational practices.

For instance, ISO/IEC/IEEE 12207 focuses on software life cycle processes and measurement activities, providing guidelines for managing software development and maintenance activities. On the other hand, ISO 9001 emphasizes quality management systems, customer satisfaction, and continual improvement of organizational processes. Understanding the differences between these standards enables organizations to identify their objectives, scope, and applications, helping them select the most suitable standard for achieving high-quality software development and effective process management. Table 5.1.6 presents a comparative overview of the key differences between ISO/IEC/IEEE 12207 and ISO 9001 standards.

SGOU

Table 5.1.6 Difference Between ISO/IEC/IEEE 12207 and ISO 9001

Feature	ISO/IEC/IEEE 12207	ISO 9001
Purpose	Defines software life cycle processes.	Defines requirements for a Quality Management System (QMS).
Primary Focus	Software development, operation, maintenance, and measurement.	Quality management, customer satisfaction, and continual improvement.
Scope	Applicable mainly to software engineering projects.	Applicable to all types of organizations and industries.
Measurement Focus	Measures software products, processes, and projects.	Measures organizational processes, products, services, and customer satisfaction.
Main Objective	Improve software development and software quality.	Improve overall organizational quality and customer satisfaction.
Key Activities	Software planning, development, testing, maintenance, and measurement.	Planning, monitoring, auditing, evaluation, and continual improvement.
Users	Software developers, project managers, software organizations.	Manufacturing, service, healthcare, education, IT, and other organizations.
Outcome	High-quality and reliable software products.	Effective Quality Management System and improved business performance.

Although ISO/IEC/IEEE 12207 and ISO 9001 have different scopes, they complement each other in achieving quality objectives. ISO/IEC/IEEE 12207 provides guidance for managing software life cycle processes, while ISO 9001 establishes a framework for quality management across the entire organization. Together, these standards help organizations deliver high-quality software, improve operational efficiency, and achieve continual improvement.

SGOU

Recap

- Software measurement is the process of collecting and analyzing data about software. It helps evaluate software products, processes, and projects.
- Software measurement supports better decision-making. The main objective is to improve software quality and performance.
- Effective measurement should be accurate, reliable, and relevant.
- Measurement helps identify defects and performance issues early.
- Software metrics are used to measure different software attributes.
- Product metrics measure the quality and characteristics of software products.
- Process metrics evaluate the effectiveness of software development processes.
- Project metrics monitor project cost, effort, schedule, and progress.
- Direct measurements are obtained without calculations, such as lines of code.
- Indirect measurements are calculated from one or more direct measurements.
- Common software attributes include quality, reliability, cost, effort, and productivity.
- ISO/IEC/IEEE 12207 provides a framework for software life cycle processes.
- The measurement process in ISO/IEC/IEEE 12207 includes planning, data collection, analysis, and reporting.
- Measurement objectives should be clearly defined before collecting data.
- Measurement results help improve software development and project management.
- **ISO 9001** focuses on Quality Management Systems (QMS).
- ISO 9001 emphasizes measurement, analysis, and evaluation of organizational performance.
- Performance measurement helps monitor process effectiveness and quality objectives.
- Customer satisfaction is measured using feedback, surveys, and complaint analysis.
- Measurement data supports evidence-based decision-making.

SGOU

- Proper documentation provides evidence of measurement activities and compliance.
- Continuous improvement is achieved by analyzing measurement results and implementing corrective actions.
- ISO/IEC/IEEE 12207 focuses on software life cycle measurement, while ISO 9001 focuses on overall quality management and continual improvement.

Objective Type Questions

1. What is the process of collecting software-related data called?
2. Which standard defines the software life cycle processes?
3. Which standard specifies Quality Management Systems?
4. What is the numerical measure used to evaluate software called?
5. Which metric measures software quality characteristics?
6. Which metric measures software development activities?
7. Which metric measures project progress?
8. What type of measurement directly counts lines of code?
9. What type of measurement is calculated from other measurements?
10. Which ISO standard focuses on customer satisfaction?
11. Which process involves collecting measurement data?
12. What is the first activity in the measurement process?
13. What is the activity of studying collected data called?
14. What is the final activity after data analysis?
15. Which document records measurement results?
16. Which audit is conducted within the organization?
17. What is the main focus of ISO 9001?
18. What is the main goal of software measurement?

SGOU

19. Which attribute indicates software correctness?

20. Which attribute measures software dependability?

21. Which activity identifies customer opinions?

22. Which process improves quality continuously?

23. What is the evidence used to prove compliance called?

24. Which management system is defined by ISO 9001?

25. Which process monitors software project performance?

Answers to Objective Type Questions

1. Measurement
2. ISO/IEC/IEEE12207
3. ISO9001
4. Metric
5. Product
6. Process
7. Project
8. Direct
9. Indirect
10. ISO9001
11. Collection
12. Planning
13. Analysis
14. Reporting
15. Documentation
16. Internal

SGOU

17. Quality

18. Improvement

19. Correctness

20. Reliability

21. Feedback

22. Improvement

23. Documentation

24. QMS

25. Monitoring

Assignments

1. Explain the concept, objectives, and importance of software measurement. Discuss the characteristics of effective software measurement with suitable examples.

2. Describe the different types of software metrics, including product metrics, process metrics, and project metrics. Differentiate between direct and indirect measurements.

3. Explain the software measurement process according to ISO/IEC/IEEE 12207. Discuss its objectives, activities, and benefits in software development.

4. Discuss the measurement requirements of ISO 9001. Explain the role of performance measurement, customer satisfaction, and continual improvement in a Quality Management System (QMS).

5. Compare ISO/IEC/IEEE 12207 and ISO 9001 with respect to their objectives, scope, measurement approach, and applications in software quality management.

SGOU

Suggested Reading

1. ISO/IEC/IEEE 12207:2017 – Systems and Software Engineering – Software Life Cycle Processes https://www.iso.org/standard/63712.html

2. ISO 9001:2015 – Quality Management Systems – Requirements https://www.iso.org/standard/62085.html

3. Software Engineering Institute (SEI), Carnegie Mellon University – Software Measurement and Process Improvement https://www.sei.cmu.edu

4. IEEE Computer Society – Software and Systems Engineering Standards https://www.computer.org/technical-committees/software-engineering

5. IBM – Software Quality Management and Metrics https://www.ibm.com/topics/software-quality

Reference

1. Pressman, R. S., & Maxim, B. R. (2024). *Software engineering: A practitioner's approach* (10th ed.). McGraw Hill.

2. Sommerville, I. (2021). *Software engineering* (10th ed.). Pearson.

3. Bourque, P., & Fairley, R. E. (Eds.). (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK® Guide)* (Version 3.0). IEEE Computer Society.

4. Hoyle, D. (2017). *ISO 9001:2015 Quality management systems handbook* (7th ed.). Routledge.

5. International Organization for Standardization. (2015). *ISO 9001:2015 Quality management systems Requirements*. ISO.

6. Towle, B. A., Jr., & Foster, E. C. (2022). *Software engineering: A methodical approach* (2nd ed.). CRC Press.

SGOU

UNIT 2

Software Measurement Models and Methods

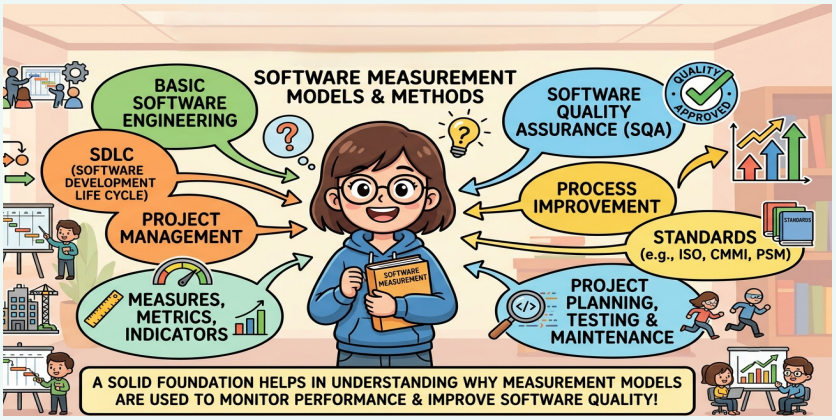
Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concept and importance of software measurement models and methods.
- describe the Practical Software and Systems Measurement (PSM) method.
- explain the measurement practices of the CMMI framework.
- identify measurement techniques used in Very Small Entities (VSEs).
- compare PSM, CMMI, and VSE measurement approaches.

Prerequisite

A basic understanding of software engineering, the Software Development Life Cycle (SDLC), and the role of software measurement in software projects is essential before studying this unit. Familiarity with concepts such as software processes, software quality, project management, measures, metrics, and indicators provides a strong foundation for understanding software measurement models and methods. Knowledge of how software projects are planned, developed, tested, and maintained helps in understanding why measurement models are used to monitor project performance and improve software quality. For example, if a software project experiences delays, metrics such as schedule variance or task completion percentage can be used to evaluate project progress.



SGOU

A basic knowledge of software quality assurance, project planning, and process improvement is also beneficial for understanding this unit. Familiarity with standards such as ISO/IEC 12207, ISO 9001, and fundamental project management practices provides useful background for studying advanced measurement models such as the Practical Software and Systems Measurement (PSM) method, the Capability Maturity Model Integration (CMMI) framework, and ISO/IEC 29110 for Very Small Entities (VSEs). For instance, a small software company may monitor defect count, test coverage, and customer satisfaction to evaluate project performance, improve software quality, and support continuous process improvement.

Keywords

PSM, CMMI, VSEs, ISO/IEC 12207, ISO 9001, ISO/IEC 2911

Discussion

5.2.1 Introduction to Software Measurement Models and Methods

Software development has evolved into a complex engineering discipline involving numerous activities such as requirement analysis, system design, coding, testing, deployment, and maintenance. As software projects become larger and more sophisticated, organizations require objective methods to monitor progress, evaluate quality, control costs, and improve development processes. Decisions based solely on intuition or personal experience often lead to inaccurate estimations, project delays, budget overruns, and poor-quality software.

Software measurement provides a systematic approach to collecting, analyzing, and interpreting quantitative data related to software products, processes, and projects. However, merely collecting data is not sufficient. Organizations need structured frameworks that define what should be measured, how measurements should be performed, and how the results should be used for decision-making. These structured frameworks are known as software measurement models and methods.

Software measurement models provide standardized guidelines for selecting metrics, collecting measurement data, analyzing results, and continuously improving software development activities. Examples include the Practical Software and Systems Measurement (PSM) method, the Capability Maturity Model Integration (CMMI) Measurement and Analysis process area, and measurement practices designed for Very Small Entities (VSEs) under ISO/IEC 29110.

By adopting appropriate measurement models, organizations can improve project planning, enhance software quality, reduce development risks, optimize resource utilization, and support continuous process improvement.

SGOU

5.2.1.1 Benefits and Challenges of Software Measurement Models

Software measurement models help organizations systematically measure and evaluate software products, processes, and projects, leading to improved quality and better project management. However, implementing these models also presents certain challenges.

Benefits of Software Measurement Models

Software measurement models provide several advantages to organizations, including:

- **Improved Decision-Making:** They provide reliable data that supports informed managerial decisions.
- **Better Project Planning:** Accurate measurements help estimate project cost, effort, schedule, and resource requirements.
- **Enhanced Software Quality:** Metrics help identify defects, improve reliability, and ensure software meets quality standards.
- **Effective Project Monitoring:** Managers can track project progress, detect delays, and take corrective actions when necessary.
- **Risk Management:** Measurement helps identify potential risks early, reducing the chances of project failure.
- **Process Improvement:** Performance data enables organizations to identify weak areas and continuously improve software development processes.
- **Increased Productivity:** Measuring team performance and process efficiency helps optimize resource utilization.
- **Customer Satisfaction:** High-quality software delivered on time improves customer confidence and satisfaction.

Challenges of Software Measurement Models

Despite their benefits, software measurement models face several challenges:

- **Selection of Appropriate Metrics:** Choosing irrelevant or excessive metrics may lead to inaccurate conclusions.
- **High Implementation Cost:** Setting up measurement programs often requires investment in tools, training, and skilled personnel.
- **Data Collection Difficulties:** Collecting consistent and accurate measurement data can be time-consuming.
- **Resistance to Measurement:** Team members may view measurement as a tool for performance evaluation rather than process improvement.
- **Data Interpretation Issues:** Incorrect analysis or misunderstanding of measurement results can lead to poor decisions.

SGOU

- **Maintenance of Measurement Programs:** Measurement models must be regularly updated to keep pace with changing technologies and development practices.

Software measurement models play a vital role in improving software quality, project management, and organizational performance. While they offer numerous benefits such as better planning, quality assurance, and continuous improvement, their successful implementation depends on selecting appropriate metrics, ensuring accurate data collection, and effectively interpreting the measurement results.

5.2.2 The Practical Software and Systems Measurement (PSM) Method

The Practical Software and Systems Measurement (PSM) method is a structured measurement framework that helps organizations effectively manage software and system development projects through the use of meaningful and actionable measurements. It was developed to provide a practical approach for planning, collecting, analyzing, and using measurement data to support project management and improve decision-making throughout the software development life cycle.

Unlike traditional measurement approaches that focus on collecting large amounts of data, the PSM method emphasizes measuring only the information that is necessary to address specific project objectives and management concerns. It follows the principle of "measure what matters", ensuring that measurements are directly linked to project goals, risks, and stakeholder information needs. This makes the PSM method practical, efficient, and adaptable to projects of different sizes and complexities.

The PSM method integrates measurement activities into everyday project management rather than treating them as separate tasks. It helps project managers monitor project progress, evaluate software quality, identify potential risks, control project costs and schedules, and make informed decisions based on objective evidence. The framework encourages continuous monitoring and improvement by regularly collecting and analyzing measurement data throughout the project.

The PSM method has been widely adopted by government agencies, defense organizations, and software industries because it provides standardized guidelines for selecting appropriate measures, interpreting results, and communicating measurement information to stakeholders. It can be applied to both software engineering projects and broader systems engineering projects, making it a flexible and comprehensive measurement approach.

5.2.2.1 Objectives and Principles of PSM

The Practical Software and Systems Measurement (PSM) method is designed to help organizations manage software and systems projects by providing meaningful measurement information for effective decision-making. Rather than collecting excessive data, PSM focuses on measuring only the information that supports project

SGOU

objectives, identifies risks, and improves project performance. Its objectives and guiding principles ensure that measurement activities remain practical, relevant, and aligned with organizational goals.

Objectives of the PSM Method

The main objectives of the Practical Software and Systems Measurement (PSM) method are:

1. **Support Decision-Making:** Provide accurate and reliable measurement data to assist project managers and stakeholders in making informed decisions.

2. **Monitor Project Progress:** Track project performance by measuring schedule, cost, effort, quality, and resource utilization.

3. **Identify and Manage Risks:** Detect potential project risks and issues early through continuous measurement and monitoring.

4. **Improve Software Quality:** Measure quality attributes such as defects, reliability, maintainability, and customer satisfaction to enhance software quality.

5. **Enhance Project Planning and Control:** Provide measurement information that supports better estimation, planning, and project management.

6. **Support Continuous Improvement:** Use measurement results to identify weaknesses and improve software development processes over time.

The PSM method aims to provide relevant and reliable measurement information that supports project planning, monitoring, risk management, and quality improvement. Its guiding principles emphasize measuring only what is necessary, integrating measurement into project management, using objective data, and continuously improving software processes. By following these objectives and principles, organizations can make better decisions, reduce project risks, and successfully deliver high-quality software systems.

5.2.2.2 PSM Framework

The Practical Software and Systems Measurement (PSM) Framework is a structured approach that guides organizations in planning, collecting, analyzing, and using measurement data throughout the software development life cycle. It ensures that measurements are directly linked to project objectives and stakeholder information needs. Instead of measuring everything, the PSM framework focuses on collecting only the information required to support effective project management and decision-making.

The framework follows a continuous cycle in which project goals are identified, appropriate measures are selected, data is collected and analyzed, and the results are used to monitor project performance and improve future decisions. This systematic approach helps organizations manage project cost, schedule, quality, risks, and resources effectively (Fig 5.2.1).

SGOU

Components of the PSM Framework

The PSM Framework consists of the following key components:

1. **Information Needs:** The first step is to identify the information required by project managers and stakeholders for effective decision-making.

Examples include:

- Is the project on schedule?
- Is the software meeting quality standards?
- Are project risks increasing?
- Is the project within budget?

2. **Measurement Objectives:** Based on the identified information needs, clear measurement objectives are established. These objectives define what should be measured and why it should be measured.

Examples include:

- Monitor project progress.
- Measure software quality.
- Track development effort.
- Assess project risks.

3. **Measures and Metrics:** Appropriate measures and metrics are selected to satisfy the measurement objectives. Examples shown in Table 5.2.1.

Table 5.2.1 Example for Measures and Metrics

Measurement Objective	Possible Metric
Project progress	Percentage of tasks completed
Software quality	Defect density
Schedule performance	Schedule variance
Cost performance	Cost variance
Testing effectiveness	Test coverage

4. **Data Collection:** Relevant measurement data is collected from various project activities using manual methods or automated tools.

SGOU

Common data sources include:

- Project management tools
- Source code repositories
- Testing tools
- Issue tracking systems
- Customer feedback

5. **Data Analysis:** The collected data is analyzed to identify trends, evaluate project performance, detect problems, and assess risks.

Typical analyses include:

- Trend analysis
- Performance comparison
- Risk analysis
- Quality assessment

6. **Reporting and Communication:** The analyzed measurement results are presented to stakeholders in a clear and understandable format.

Common reporting methods include:

- Dashboards
- Graphs and charts
- Performance reports
- Risk reports
- Progress summaries

7. **Decision-Making and Improvement:** The final step is to use the measurement results for project management and continuous improvement.

Managers use the information to:

- Resolve project issues
- Improve software quality
- Adjust schedules
- Allocate resources efficiently
- Reduce project risks
- Improve future projects

SGOU

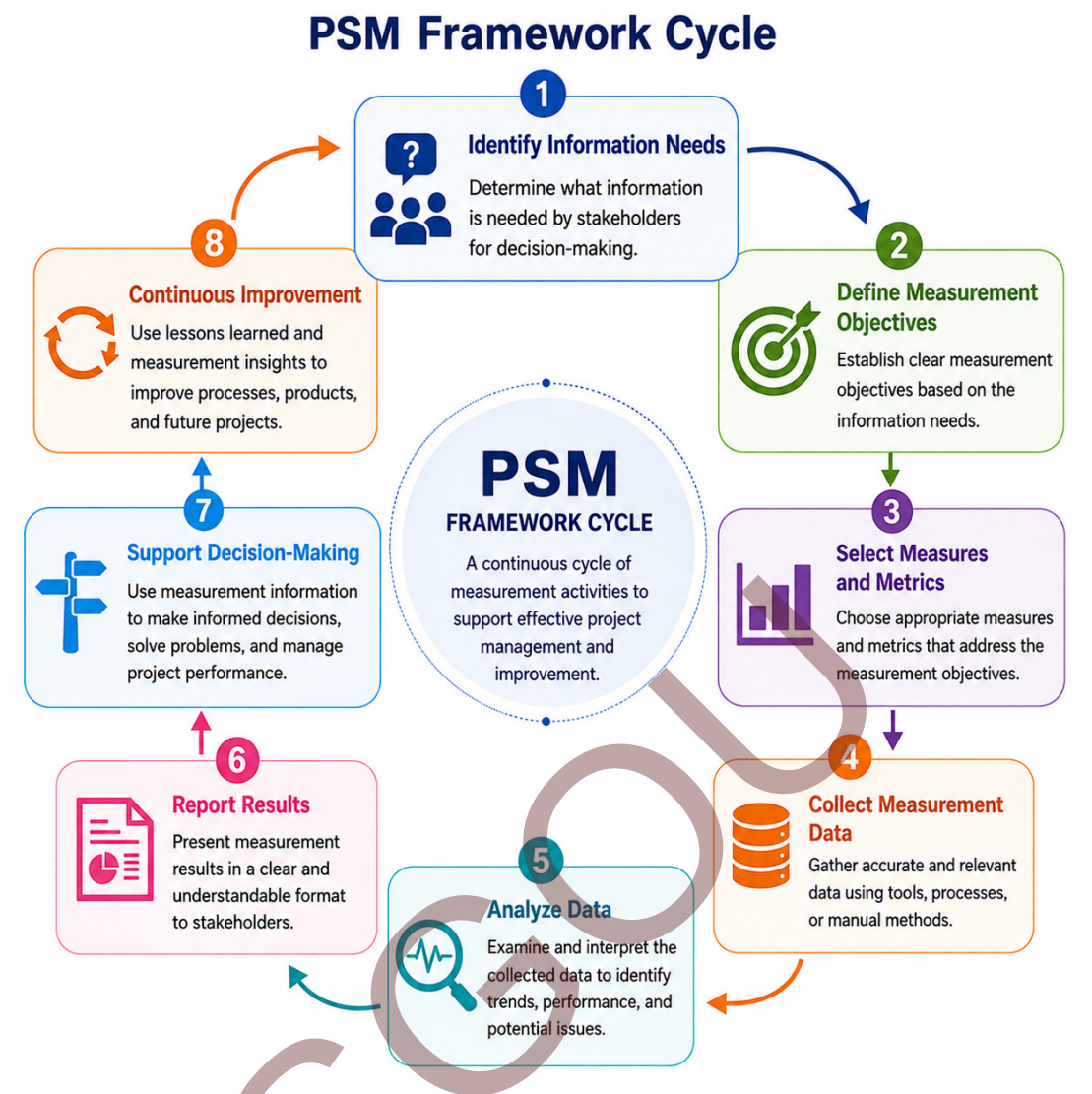


Fig 5.2.1 PSM Measurement Framework Cycle

The PSM Framework provides a simple and systematic approach to software measurement by aligning measurement activities with project goals and stakeholder needs. It helps organizations collect, analyze, and use measurement data to improve project performance, reduce risks, and support continuous improvement.

5.2.2.3 Information Needs and Measurement Planning

Information needs and measurement planning is an important step in the Practical Software and Systems Measurement (PSM) method. It focuses on identifying the information required by project stakeholders and planning how that information will be measured. Instead of collecting unnecessary data, PSM begins by asking what information is needed for effective decision-making. Once the information needs are identified, an appropriate measurement plan is developed to specify what will be measured, how the data will be collected, who will collect it, and how the results will be analyzed and reported.

SGOU

A well-defined measurement plan ensures that measurement activities are aligned with project goals and stakeholder expectations. It also helps organizations collect relevant, accurate, and timely data, enabling project managers to monitor project performance, identify risks, and take corrective actions when necessary.

Information Needs

Information needs refer to the specific information required by project managers, developers, customers, and other stakeholders to make informed decisions and successfully manage the project. Common information needs include:

- Is the project progressing according to schedule?
- Is the project within the planned budget?
- Does the software meet quality requirements?
- Are there any significant project risks?
- Are resources being used efficiently?
- Is customer satisfaction improving?

These questions help determine what measurements are necessary for the project.

Measurement Planning

Measurement planning is the process of defining how measurement activities will be carried out during the project. It ensures that the right data is collected systematically and used effectively. A typical measurement plan includes:

- **Measurement Objectives:** Define the purpose of measurement.
- **Measures and Metrics:** Identify the metrics to be collected.
- **Data Sources:** Specify where the measurement data will be obtained.
- **Data Collection Methods:** Determine how data will be collected (manual or automated).
- **Roles and Responsibilities:** Assign responsibility for collecting, analyzing, and reporting data.
- **Measurement Schedule:** Decide when and how often measurements will be taken.
- **Reporting Method:** Define how measurement results will be communicated to stakeholders.

Steps in Information Needs and Measurement Planning

1. Identify stakeholder information needs.
2. Define measurement objectives.
3. Select appropriate measures and metrics.

SGOU

4. Identify data sources.
5. Develop data collection procedures.
6. Analyze and interpret measurement data.
7. Report measurement results.
8. Review and improve the measurement plan regularly.

Example: Consider a software development project where the project manager wants to ensure timely completion and high-quality software (shown in Table 5.2.2).

Table 5.2.2 Example for Information Needs and Measurement Planning

Information Need	Measurement
Is the project on schedule?	Schedule variance, task completion percentage
Is the software quality improving?	Defect density, test coverage
Are project costs under control?	Cost variance, budget utilization
Are risks increasing?	Number of open risks, risk severity

Information needs and measurement planning are the first steps in the PSM method, ensuring that only relevant information is measured. They help organizations collect useful data, monitor project performance, manage risks, and make informed decisions throughout the software development life cycle.

5.2.2.4 Selecting and Defining Measures

Selecting and defining measures is a key activity in the Practical Software and Systems Measurement (PSM) method. After identifying the information needs and measurement objectives, the next step is to choose appropriate measures that provide meaningful and reliable information. The selected measures should be directly related to project goals, easy to collect, and useful for monitoring project performance and supporting decision-making.

Defining measures involves clearly specifying what will be measured, how the measurement will be performed, the units of measurement, the data source, and the frequency of data collection. Well-defined measures ensure consistency, accuracy, and comparability of measurement data throughout the software development life cycle.

Criteria for Selecting Measures

The measures selected should meet the following criteria:

- **Relevant:** Directly support project objectives and stakeholder information needs.
- **Measurable:** Data should be quantifiable and easy to collect.

SGOU

- **Reliable:** Measurements should produce consistent results.
- **Simple:** Measures should be easy to understand and interpret.
- **Cost-Effective:** Data collection should not require excessive time or resources.
- **Timely:** Information should be available when needed for decision-making.
- **Actionable:** Results should help managers take appropriate corrective actions.

Defining Measures

Each measure should be clearly defined to avoid confusion and ensure consistent data collection (Table 5.2.3). A measurement definition typically includes:

- **Measure Name:** The name of the metric or measure.
- **Purpose:** Why the measure is being collected.
- **Description:** What is being measured.
- **Unit of Measurement:** Hours, percentage, number of defects, etc.
- **Data Source:** Project documents, testing tools, issue trackers, or code repositories.
- **Collection Method:** Manual or automated.
- **Collection Frequency:** Daily, weekly, monthly, or at project milestones.
- **Responsible Person:** Individual or team responsible for collecting and maintaining the data.

Table 5.2.3 Examples of Common Measures

Measurement Objective	Selected Measure
Monitor project progress	Percentage of tasks completed
Measure software quality	Defect density
Evaluate testing	Test coverage (%)
Monitor schedule	Schedule variance
Control project cost	Cost variance
Measure productivity	Function points per developer
Assess customer satisfaction	Customer satisfaction score

SGOU

Steps for Selecting and Defining Measures

1. Identify project objectives and stakeholder information needs.
2. Select measures that support these objectives.
3. Clearly define each measure and its purpose.
4. Specify the data source and collection method.
5. Establish measurement frequency.
6. Assign responsibility for data collection.
7. Review measures regularly to ensure they remain useful and relevant.

Example: A software company wants to improve product quality.

- **Objective:** Reduce software defects.
- **Selected Measure:** Defect density (number of defects per thousand lines of code).
- **Data Source:** Bug tracking system.
- **Collection Frequency:** Weekly.
- **Responsible Person:** Quality Assurance (QA) team.

The collected data is analyzed regularly to determine whether software quality is improving and whether corrective actions are required.

Selecting and defining measures is an important step in the PSM method that ensures appropriate metrics are chosen to meet project objectives. Clearly defined and relevant measures help organizations collect accurate data, monitor project performance, and support better decision-making and continuous improvement.

5.2.2.5 Data Collection, Analysis, and Reporting

Data collection, analysis, and reporting are essential activities in the Practical Software and Systems Measurement (PSM) method. Once appropriate measures and metrics have been selected, the required data is collected from various sources, analyzed to evaluate project performance, and reported to stakeholders. These activities ensure that project managers have accurate and timely information to monitor progress, identify risks, improve software quality, and make informed decisions.

The PSM method emphasizes collecting only relevant and reliable data that supports project objectives. The collected data is transformed into meaningful information through analysis and then presented in a clear and understandable format to facilitate effective project management.

SGOU

1. Data Collection

Data collection is the process of gathering measurement data from software development activities. The collected data should be accurate, consistent, and relevant to the project's objectives. Table 5.2.4 lists the common sources of software measurement data, the methods used to collect it, and examples of measurement data.

Table 5.2.4 Software Measurement Data Sources, Collection Methods, and Examples

Common Sources of Data	Methods of Data Collection	Examples of Collected Data
Project management tools	Manual data collection	Number of defects
Source code repositories	Automated data collection using software tools	Development effort (hours)
Bug tracking systems	Surveys and questionnaires	Project cost
Testing tools	System logs and monitoring tools	Test coverage
Version control systems		Task completion percentage
Customer feedback		Customer satisfaction ratings
Project documentation		Change requests (if applicable)

2. Data Analysis

Data analysis is the process of examining the collected data to identify trends, evaluate performance, and detect potential problems. The analysis converts raw data into meaningful information that supports project management and decision-making. Table 5.2.5 summarizes the common data analysis techniques and their role in supporting software project management and process improvement.

Table 5.2.5 Common Analysis Techniques and Purpose of Data Analysis

Common Analysis Techniques	Purpose of Data Analysis
Trend analysis	Monitor project progress
Performance comparison	Evaluate software quality
Defect analysis	Identify project risks

SGOU

Risk analysis	Detect performance issues
Statistical analysis	Compare planned and actual results
Quality assessment	Support continuous improvement

3. Reporting

Reporting is the process of presenting the analyzed measurement results to stakeholders in a clear, concise, and understandable format. Effective reports enable project managers, developers, and customers to understand project status and make informed decisions. Table 5.2.6 illustrates the common reporting methods used in software measurement and the key characteristics of effective measurement reports.

Table 5.2.6 Common Reporting Methods and Characteristics of Effective Reports

Common Reporting Methods	Characteristics of Effective Reports
Dashboards	Accurate and reliable
Progress reports	Easy to understand
Graphs and charts	Relevant to stakeholder needs
Performance reports	Timely and up-to-date
Risk reports	Visually clear using tables, charts, and graphs
Executive summaries	Concise and actionable

Example: A software development team collects weekly project data (Table 5.2.7).

Table 5.2.7: Example for Data Collection, Analysis, and Reporting

Activity	Example
Data Collection	Number of defects reported, hours worked, test cases executed
Data Analysis	Compare planned and actual progress, calculate defect density
Reporting	Weekly project status report with charts and performance indicators

The project manager reviews the report to determine whether the project is progressing according to schedule and whether corrective actions are required.

Data collection, analysis, and reporting are key activities in the PSM method that convert raw data into useful information. They help organizations monitor project progress, improve software quality, manage risks, and make informed decisions throughout the software development life cycle.

SGOU

5.2.2.7 Advantages and Limitations of the PSM Method

The Practical Software and Systems Measurement (PSM) method is a widely used measurement framework that helps organizations manage software and systems projects through systematic measurement. It provides a structured approach for selecting meaningful measures, collecting and analyzing data, and using the results to support project management. While the PSM method offers several benefits, it also has certain limitations that organizations should consider during implementation.

Advantages of the PSM Method

The PSM method provides the following advantages:

1. **Supports Better Decision-Making:** Provides reliable and objective measurement data for informed project decisions.

2. **Improves Project Monitoring:** Enables continuous tracking of project progress, cost, schedule, quality, and risks.

3. **Focuses on Relevant Measurements:** Measures only the information that is necessary to achieve project objectives, avoiding unnecessary data collection.

4. **Enhances Software Quality:** Helps identify defects and quality issues early, leading to improved software reliability and performance.

5. **Supports Risk Management:** Detects potential project risks at an early stage, allowing timely corrective actions.

6. **Integrates with Project Management:** Measurement activities are incorporated into regular project management rather than treated as separate tasks.

7. **Promotes Continuous Improvement:** Measurement results help organizations improve software development processes and project performance over time.

8. **Flexible and Scalable:** Can be applied to projects of different sizes, complexities, and application domains.

Limitations of the PSM Method

Despite its advantages, the PSM method has some limitations:

1. **Requires Skilled Personnel:** Successful implementation requires personnel with knowledge of software measurement and data analysis.

2. **Time-Consuming Implementation:** Planning, collecting, and analyzing measurement data may require considerable time and effort.

3. **Dependence on Data Quality:** Inaccurate or incomplete data can lead to incorrect conclusions and poor decision-making.

SGOU

4. **Initial Implementation Cost:** Organizations may need to invest in measurement tools, training, and process improvements.

5. **Resistance to Measurement:** Team members may resist measurement activities if they perceive them as performance evaluation rather than process improvement.

6. **Maintenance of Measurement Program:** Measures and metrics must be reviewed and updated regularly to remain relevant as project requirements change.

7. **Not a Complete Project Management Solution:** PSM provides measurement support but does not replace project management methodologies such as Agile or Scrum.

8. **May Require Additional Tools and Resources:** The PSM method may require additional software tools, skilled personnel, and resources for effective measurement, increasing implementation cost and effort.

The Practical Software and Systems Measurement (PSM) method provides a structured approach to measuring and managing software projects. It helps improve project performance, software quality, decision-making, and continuous improvement through effective use of measurement data.

5.2.3 Measurement According to the Capability Maturity Model Integration (CMMI)

The Capability Maturity Model Integration (CMMI) is a globally recognized framework that helps organizations improve the quality and efficiency of their software and systems development processes. Within CMMI, measurement is considered a critical activity for understanding project performance, evaluating process effectiveness, and supporting continuous improvement. The Measurement and Analysis (MA) process area in CMMI provides guidelines for defining measurement objectives, selecting appropriate metrics, collecting reliable data, and analyzing the results to support informed decision-making. By using objective measurement data, organizations can monitor project progress, identify potential risks, and ensure that development activities align with business goals.

Measurement in CMMI is integrated into everyday project management rather than being treated as a separate activity. Organizations establish measurement goals based on stakeholder information needs, collect data using standardized procedures, analyze the results to identify trends and performance issues, and communicate the findings through reports and dashboards. This systematic approach helps improve software quality, project planning, resource utilization, and customer satisfaction while promoting continuous process improvement. As organizations advance through the CMMI maturity levels, measurement practices become more consistent, data-driven, and effective in achieving organizational objectives.

SGOU

5.2.3.1 CMMI Framework

The Capability Maturity Model Integration (CMMI) is a process improvement framework that helps organizations develop high-quality software and systems by improving their development and management processes. Developed by the Software Engineering Institute (SEI) at Carnegie Mellon University, CMMI provides a structured set of best practices that enable organizations to improve productivity, product quality, customer satisfaction, and project performance. It can be applied to software engineering, systems engineering, product development, service management, and supplier management across organizations of all sizes.

The CMMI framework emphasizes continuous process improvement through the use of standardized practices, measurement, and performance evaluation. It encourages organizations to define and manage processes systematically, monitor project performance using measurable objectives, identify areas for improvement, and implement corrective actions. By adopting CMMI, organizations can reduce project risks, improve estimation accuracy, enhance software quality, and achieve consistent project outcomes.

Key Components of the CMMI Framework

The CMMI framework consists of several process areas that define best practices for software and systems development. One of the important process areas is Measurement and Analysis (MA), which helps organizations establish measurement objectives, collect and analyze data, and use the results to support project management and process improvement (as discussed thoroughly in preceding units).

CMMI Maturity Levels

The CMMI framework defines five maturity levels (Table 5.2.8) that represent the capability of an organization's processes :

Table 5.2.8 CMMI Maturity Levels

Maturity Level	Description
Level 1 – Initial	Processes are unpredictable, informal, and reactive. Success depends on individual effort.
Level 2 – Managed	Basic project management processes are established. Projects are planned, monitored, and measured.
Level 3 – Defined	Standardized processes are documented and consistently followed across the organization.
Level 4 – Quantitatively Managed	Processes are measured and controlled using quantitative data and statistical techniques.
Level 5 – Optimizing	Continuous process improvement is achieved through innovation, measurement, and performance analysis.

The CMMI Framework is a process improvement model that helps organizations improve the quality, efficiency, and reliability of software development. By following

SGOU

standardized practices and using measurement, organizations can reduce risks, improve software quality, and achieve continuous process improvement.

5.2.3.2 Measurement and Analysis (MA) Process Area

The Measurement and Analysis (MA) Process Area is one of the key process areas in the Capability Maturity Model Integration (CMMI) framework. It provides a structured approach for collecting, analyzing, storing, and reporting measurement data to support effective project management and process improvement. The primary purpose of the MA process area is to ensure that organizations collect meaningful and reliable data that meets the information needs of project managers, stakeholders, and decision-makers. By using objective measurement data, organizations can monitor project performance, assess software quality, identify risks, and make informed decisions throughout the software development life cycle.

The MA process area emphasizes that measurement activities should be planned, systematic, and aligned with organizational goals. It begins by defining measurement objectives based on business and project needs, followed by selecting appropriate measures and metrics. Data is then collected from reliable sources, analyzed to identify trends and performance issues, and communicated to stakeholders through reports and dashboards. The analyzed results are used to evaluate project progress, improve development processes, and support continuous process improvement.

The CMMI Measurement and Analysis process area consists of two specific goals. Table 5.2.9 lists the specific goals and practices of the CMMI Measurement and Analysis (MA) process area for planning, managing, and reporting software measurement activities.

Table 5.2.9 Specific Goals and Practices of the CMMI Measurement and Analysis (MA) Process Area

Specific Goal (SG)	Specific Practices (SP)
SG 1: Align Measurement and Analysis Activities	♦ Establish measurement objectives. ♦ Specify appropriate measures and metrics. ♦ Define data collection and storage procedures. ♦ Establish analysis methods.
SG 2: Provide Measurement Results	♦ Collect measurement data. ♦ Analyze and interpret the collected data. ♦ Store measurement information securely. ♦ Report measurement results to stakeholders.

The Measurement and Analysis (MA) Process Area is an important part of the CMMI framework that helps organizations measure and analyze project and process

SGOU

performance. It supports informed decision-making, improves software quality, enhances project management, and promotes continuous process improvement.

5.2.3.3 Specifying Measures and Metrics

Specifying Measures and Metrics is an important activity in the Measurement and Analysis (MA) process area of the Capability Maturity Model Integration (CMMI) framework. After establishing measurement objectives, organizations identify and define the measures and metrics that will be used to evaluate project and process performance. The selected measures should be directly aligned with organizational goals, stakeholder information needs, and project objectives. Clearly defined measures ensure that the collected data is accurate, consistent, and useful for monitoring progress and supporting informed decision-making.

A measure refers to the raw data collected about a software product, process, or project, while a metric is a calculated value derived from one or more measures that provides meaningful insights into performance. For example, the number of defects reported is a measure, whereas defect density (defects per thousand lines of code) is a metric. In the CMMI framework, each measure and metric should have a clear definition, including its purpose, method of calculation, data source, collection frequency, and responsible personnel. This standardization ensures consistency in data collection and enables meaningful comparison across projects.

Guidelines for Specifying Measures and Metrics

- Align measures with business and project objectives.
- Select relevant and meaningful metrics.
- Ensure measures are simple, reliable, and easy to collect.
- Clearly define each measure and its calculation method.
- Specify data sources and collection procedures.
- Determine the frequency of data collection and reporting.
- Assign responsibility for collecting and maintaining measurement data.
- Periodically review and update measures to ensure their continued relevance.

Table 5.2.10 depicts examples of commonly used measures and metrics for evaluating software project performance, quality, and productivity.

SGOU

Table 5.2.10 Examples of Measures and Metrics

Measurement Objective	Measure	Metric
Monitor project progress	Tasks completed	Task completion percentage
Improve software quality	Number of defects	Defect density
Evaluate testing	Test cases executed	Test coverage (%)
Monitor project schedule	Actual completion time	Schedule variance
Control project cost	Actual project cost	Cost variance
Improve productivity	Development hours	Function points per developer

Specifying measures and metrics is an important step in the CMMI Measurement and Analysis process area. It helps organizations collect reliable data, monitor project performance, improve software quality, and support continuous process improvement by using clearly defined and relevant measures.

5.2.3.4 Collecting, Storing, and Analyzing Measurement Data

Collecting, storing, and analyzing measurement data is a key activity in the Measurement and Analysis (MA) process area of the CMMI framework. After measures and metrics are defined, data is systematically collected from software development activities, stored securely, and analyzed to evaluate project and process performance. This ensures that managers and stakeholders have accurate information to monitor progress, identify risks, improve software quality, and make informed decisions. Regular analysis of the collected data helps identify trends, compare planned and actual performance, detect potential issues, and support continuous process improvement.

1. **Collecting Measurement Data :** Data collection involves gathering information related to software products, processes, and projects from reliable sources. Table 5.2.11 presents the common sources of software measurement data and examples of the information collected from these sources.

Table 5.2.11 Common Sources of Measurement Data and Examples of Collected Data

Common Sources of Measurement Data	Examples of Collected Data
Project management tools	Task completion percentage
Source code repositories	Development effort (hours)
Version control systems	Project cost
Bug tracking systems	Number of defects

SGOU

Testing tools	Test coverage
Customer feedback	Customer satisfaction ratings
Project documentation	Project cost, effort, and progress information

2. **Storing Measurement Data:** Once collected, the measurement data should be stored securely and systematically so that it can be accessed for analysis, reporting, and future reference.

3. **Analyzing Measurement Data:** Data analysis involves examining the collected data to evaluate project performance, identify trends, and detect potential problems.

Common Analysis Techniques

- Trend analysis
- Performance comparison
- Statistical analysis
- Defect analysis
- Risk analysis
- Quality assessment

Steps in Collecting, Storing, and Analyzing Measurement Data

1. Collect data from identified sources.
2. Verify the accuracy and completeness of the data.
3. Store the data in a secure and organized repository.
4. Analyze the data using appropriate techniques.
5. Identify trends, issues, and performance gaps.
6. Interpret the analysis results.
7. Share the findings with stakeholders.
8. Use the results for project management and continuous improvement.

Example: A software company monitors the progress of a development project by collecting data every week (Table 5.2.12).

SGOU

Table 5.2.12 Example for Collecting, Storing, and Analyzing Measurement Data

Activity	Example
Collect	Number of defects, development hours, test cases executed
Store	Save data in a centralized project database
Analyze	Calculate defect density, schedule variance, and productivity trends

The analyzed results help the project manager determine whether the project is progressing as planned and identify any corrective actions required.

Collecting, storing, and analyzing measurement data is an important activity in the CMMI Measurement and Analysis process area that helps organizations manage software projects effectively. By systematically handling measurement data, organizations can monitor performance, improve software quality, manage risks, support informed decision-making, and achieve continuous process improvement.

5.2.3.5 Reporting and Communicating Measurement Results

Reporting and Communicating Measurement Results is the final activity in the Measurement and Analysis (MA) process area of the CMMI framework. After measurement data is collected, stored, and analyzed, the results are presented to project managers and other stakeholders through reports, dashboards, charts, and performance summaries. Effective reporting helps monitor project status, compare actual performance with planned objectives, identify risks and trends, support informed decision-making, and promote transparency, accountability, and continuous process improvement.

Methods of Reporting Measurement Results

Organizations communicate measurement results using various methods, including:

- Project status reports
- Dashboards
- Graphs and charts
- Performance reports
- Risk reports
- Executive summaries
- Review meetings and presentations

Example: A software development team prepares a monthly project performance report (Table 5.2.13).

SGOU

Table 5.2.13 Example for Measurement Results and Reported Information

Measurement Result	Reported Information
Project progress	80% of planned tasks completed
Software quality	Defect density reduced by 15%
Schedule performance	Project is one week ahead of schedule
Testing	95% of test cases successfully executed

The project manager shares this report with stakeholders to review project status, discuss any issues, and plan necessary corrective actions.

Reporting and communicating measurement results is an important activity in the CMMI Measurement and Analysis process area. It helps stakeholders understand project performance, make informed decisions, improve software quality, and support continuous process improvement.

5.2.3.6 Benefits and limitations of CMMI

The Capability Maturity Model Integration (CMMI) framework is a widely recognized process improvement model that helps organizations enhance the quality, efficiency, and predictability of software and systems development. By implementing standardized processes and measurement practices, organizations can improve project performance and achieve continuous improvement. However, implementing CMMI also presents certain challenges, especially for small organizations with limited resources.

Benefits of the CMMI Framework

The CMMI framework offers the following benefits:

1. **Improves Software Quality:** Standardized processes help reduce defects and improve the reliability and quality of software products.
2. **Enhances Project Management:** Supports better planning, monitoring, and control of project cost, schedule, and resources.
3. **Supports Data-Driven Decision-Making:** Uses measurement and analysis to provide objective information for effective decision-making.
4. **Increases Process Efficiency:** Well-defined processes improve productivity and reduce unnecessary work.
5. **Reduces Project Risks:** Early identification of risks enables timely corrective actions and minimizes project failures.
6. **Promotes Continuous Process Improvement:** Encourages organizations to regularly evaluate and improve their development processes.

SGOU

7. **Improves Customer Satisfaction:** High-quality products delivered on time increase customer confidence and satisfaction.

8. **Enhances Organizational Reputation:** CMMI certification demonstrates a commitment to quality and process excellence, improving credibility and competitiveness.

Limitations of the CMMI Framework

Despite its advantages, the CMMI framework has some limitations:

1. **High Implementation Cost:** Implementing CMMI requires investment in training, tools, process documentation, and assessments.

2. **Time-Consuming:** Establishing and maintaining CMMI processes can take considerable time and effort.

3. **Requires Skilled Personnel:** Successful implementation depends on experienced professionals who understand CMMI practices.

4. **Complex Documentation:** Extensive process documentation may increase administrative workload.

5. **Resistance to Change:** Employees may resist adopting new processes and procedures.

6. **Less Suitable for Very Small Organizations:** Small organizations with limited resources may find full CMMI implementation challenging.

7. **Requires Continuous Maintenance:** Processes and measurement practices must be regularly reviewed and updated to remain effective.

8. **Additional Resources May Be Required:** Implementing CMMI may require additional tools, training, skilled personnel, and financial resources, increasing the overall cost and effort.

The CMMI framework provides a structured approach to improving software development processes through standardized practices and measurement. It helps organizations improve software quality, manage projects effectively, reduce risks, and achieve continuous process improvement.

5.2.4 Measurement in Very Small Entities (VSEs)

Very Small Entities (VSEs) are organizations, enterprises, departments, or projects with up to 25 people involved in software development and maintenance. These organizations often have limited financial resources, fewer employees, and simpler management structures compared to large enterprises. Despite their size, VSEs face challenges similar to larger organizations, such as delivering high-quality software, meeting customer requirements, controlling project costs, and completing projects on time. Therefore, software measurement plays a vital role in helping VSEs monitor project performance, improve software quality, and make informed decisions.

SGOU

Measurement in VSEs focuses on using simple, practical, and cost-effective metrics rather than complex measurement systems. International standards such as ISO/IEC 29110 provide guidelines specifically designed for VSEs, enabling them to implement software measurement practices without requiring extensive resources or expertise. These measurement practices help organizations monitor project progress, manage risks, evaluate product quality, and continuously improve development processes while keeping measurement activities easy to implement and maintain.

Objectives of Measurement in VSEs

The primary objectives of measurement in Very Small Entities are:

- Monitor project progress and performance.
- Improve software quality.
- Support effective decision-making.
- Estimate project cost, effort, and schedule accurately.
- Identify and manage project risks.
- Improve productivity and resource utilization.
- Support continuous process improvement.
- Increase customer satisfaction.

Common metrics used in VSEs are:

- Number of defects
- Project completion percentage
- Development effort (hours)
- Schedule variance
- Cost variance
- Customer satisfaction
- Test coverage
- Number of change requests

Benefits of measurement in VSEs are:

- Improves project planning and control.
- Enhances software quality.
- Supports evidence-based decision-making.

SGOU

- Helps identify risks early.
- Improves team productivity.
- Enables continuous improvement with minimal overhead.
- Builds customer confidence through consistent project delivery.

Challenges faced by VSEs are:

- Limited financial and human resources.
- Lack of measurement expertise.
- Difficulty in adopting complex measurement frameworks.
- Time constraints for collecting and analyzing data.
- Limited access to advanced measurement tools.

Measurement in Very Small Entities (VSEs) provides a practical and efficient approach to monitoring software projects using simple and meaningful metrics. By following standards such as ISO/IEC 29110, VSEs can implement measurement practices that improve project management, software quality, and customer satisfaction without the complexity and cost associated with large-scale measurement frameworks. These practices enable small organizations to achieve continuous improvement and deliver successful software projects despite limited resources.

5.2.4.1 Need for Measurement in VSEs

Very Small Entities (VSEs) often operate with limited staff, budget, and technical resources. Despite these constraints, they must deliver high-quality software, complete projects on time, and satisfy customer requirements. Software measurement helps VSEs monitor project performance, control costs, improve software quality, and make informed decisions using objective data. By measuring key project activities, VSEs can identify problems early, use resources efficiently, and continuously improve their development processes. Simple and practical measurement practices, such as those recommended in ISO/IEC 29110, enable VSEs to achieve better project outcomes without requiring complex measurement systems. Need for measurement in VSEs are:

1. **Monitor Project Progress:** Tracks project status and ensures work is progressing according to the plan.
2. **Improve Software Quality:** Measures defects, testing progress, and product quality to deliver reliable software.
3. **Support Better Decision-Making:** Provides accurate data to help managers make informed project decisions.
4. **Manage Project Cost and Schedule:** Helps control project expenses and ensures timely completion of tasks.

SGOU

5. **Identify Risks Early:** Detects potential problems before they become critical and enables timely corrective actions.

6. **Improve Resource Utilization:** Ensures efficient use of limited human resources, budget, and development tools.

7. **Increase Productivity:** Measures team performance and identifies opportunities to improve efficiency.

8. **Enhance Customer Satisfaction:** Ensures software meets customer requirements and is delivered on time.

9. **Support Continuous Improvement:** Uses measurement results to improve software development processes and project performance.

10. **Comply with Standards:** Helps VSEs implement measurement practices recommended by standards such as ISO/IEC 29110.

Measurement is essential for Very Small Entities because it provides a simple and effective way to monitor projects, improve software quality, optimize limited resources, and support informed decision-making. By implementing practical measurement practices, VSEs can reduce risks, increase productivity, achieve continuous improvement, and deliver successful software projects within budget and schedule.

5.2.4.2 ISO/IEC 29110 and Measurement Practices

ISO/IEC 29110 is an international standard specifically developed for Very Small Entities (VSEs) involved in software and systems engineering. A Very Small Entity is an organization, department, or project with up to 25 people. The standard provides practical guidelines and best practices that help VSEs manage software development activities without the complexity of large organizational frameworks. One of the important aspects of ISO/IEC 29110 is the use of software measurement to monitor project performance, improve product quality, and support informed decision-making. The standard encourages organizations to adopt simple, effective, and cost-efficient measurement practices that are easy to implement and maintain.

Measurement practices in ISO/IEC 29110 focus on collecting only the essential information required to manage software projects successfully. Rather than using a large number of complex metrics, the standard recommends selecting a few meaningful measures related to project progress, quality, effort, cost, and customer satisfaction. The collected data is analyzed regularly to monitor performance, identify problems, and improve software development processes. By integrating measurement into everyday project activities, ISO/IEC 29110 enables VSEs to achieve better project control, continuous improvement, and higher customer satisfaction with minimal overhead.

SGOU

Objectives of ISO/IEC 29110 Measurement Practices

The measurement practices in ISO/IEC 29110 aim to:

- Monitor project progress and performance.
- Improve software product quality.
- Support informed decision-making.
- Control project cost, effort, and schedule.
- Identify project risks at an early stage.
- Improve resource utilization.
- Encourage continuous process improvement.
- Increase customer satisfaction.

Key Measurement Practices in ISO/IEC 29110

1. **Define Measurement Objectives:** Identify what information is needed to achieve project goals.
2. **Select Simple and Relevant Metrics:** Choose metrics that are easy to collect and directly support project management.
3. **Collect Measurement Data:** Gather accurate data from project activities using manual or automated methods.
4. **Analyze Measurement Results:** Evaluate the collected data to identify trends, issues, and opportunities for improvement.
5. **Report Measurement Information:** Present the results clearly to project managers and stakeholders using reports or dashboards.
6. **Use Results for Continuous Improvement:** Apply measurement findings to improve project performance, software quality, and development processes.

Table 5.2.14 shows examples of simple and practical metrics recommended for Very Small Entities (VSEs) to monitor project performance, software quality, and customer satisfaction.

SGOU

Table 5.2.14 Examples of Metrics Recommended for VSEs

Measurement Area	Example Metric
Project Progress	Percentage of tasks completed
Schedule	Schedule variance
Cost	Budget utilization
Software Quality	Number of defects, defect density
Productivity	Development effort (hours)
Testing	Test coverage
Customer Satisfaction	Customer feedback score

ISO/IEC 29110 provides a practical and lightweight approach to software measurement for Very Small Entities (VSEs). Its measurement practices emphasize simplicity, relevance, and cost-effectiveness by focusing on essential project metrics that support project management and process improvement. By adopting these practices, VSEs can monitor project performance, improve software quality, manage risks effectively, and achieve continuous improvement while making efficient use of their limited resources.

5.2.4.3 Simple Metrics for Small Software Projects

Small software projects require simple, easy-to-understand, and cost-effective metrics to monitor project performance and software quality. Unlike large organizations that use complex measurement systems, small software teams and Very Small Entities (VSEs) focus on a limited number of meaningful metrics that provide valuable information for project management and decision-making. These metrics help track project progress, control costs and schedules, identify quality issues, and improve productivity without increasing the workload.

Simple metrics are easy to collect, analyze, and interpret, making them suitable for organizations with limited resources. By regularly monitoring these metrics, project managers can identify problems early, take corrective actions, and continuously improve software development processes (Table 5.2.15).

Table 5.2.15 Common Simple Metrics for Small Software Projects

Metric	Purpose
Task Completion Percentage	Measures the percentage of planned tasks completed.
Number of Defects	Tracks the total defects identified during development and testing.
Defect Density	Measures the number of defects per unit of software (e.g., per 1,000 lines of code or function point).

SGOU

Development Effort	Measures the total hours spent on software development activities.
Schedule Variance	Compares planned completion time with the actual completion time.
Cost Variance	Compares the planned project cost with the actual cost incurred.
Test Coverage	Measures the percentage of requirements or code tested.
Customer Satisfaction	Evaluates customer feedback on the delivered software.
Change Requests	Counts the number of requested modifications after development begins.
Productivity	Measures software output, such as features completed or function points developed per developer.

Example: A small software development team developing a web application tracks the following metrics every week (Table 5.2.16).

Table 5.2.16 Example for Metrics

Metric	Current Value
Task Completion Percentage	80%
Number of Defects	12
Development Effort	240 hours
Test Coverage	92%
Customer Satisfaction	4.6/5

Based on these metrics, the project manager can evaluate project progress, identify areas needing improvement, and take corrective actions to ensure successful project completion.

Simple metrics help small software projects and Very Small Entities (VSEs) monitor project performance in an easy and practical way. By tracking metrics such as task completion, defects, effort, cost, and customer satisfaction, organizations can improve software quality, manage projects effectively, and support continuous improvement.

5.2.4.4 Cost-Effective Measurement Techniques

Cost-effective measurement techniques are simple and practical methods that help Very Small Entities (VSEs) and small software development teams monitor project performance with minimal time, cost, and resources. These techniques focus on collecting only essential data to support project planning, quality improvement, risk

SGOU

management, and decision-making, allowing organizations to improve software quality and project performance without using expensive tools or complex measurement frameworks. Cost-effective measurement techniques are:

1. **Measure Only Essential Metrics:** Collect only the most important metrics, such as project progress, defects, effort, schedule, and customer satisfaction.

2. **Use Simple Measurement Methods:** Prefer easy-to-understand measures that can be collected and interpreted without complex calculations.

3. **Automate Data Collection:** Use project management, version control, and issue-tracking tools to automatically collect measurement data whenever possible.

4. **Use Free or Open-Source Tools:** Utilize free software tools for project tracking, bug reporting, testing, and reporting to reduce implementation costs.

5. **Integrate Measurement with Daily Activities:** Collect measurement data as part of routine development activities instead of performing separate measurement tasks.

6. **Reuse Existing Project Data:** Use information already available in project documents, code repositories, and testing reports instead of collecting new data.

7. **Review Metrics Regularly:** Periodically evaluate the usefulness of metrics and remove unnecessary measurements to reduce effort.

8. **Focus on Continuous Improvement:** Use measurement results to identify improvement opportunities rather than simply collecting data.

Table 5.2.17 summarizes examples of software metrics that provide valuable project information while minimizing measurement costs.

Table 5.2.17 Examples of Cost-Effective Metrics

Metric	Purpose
Task Completion Percentage	Monitor project progress
Number of Defects	Evaluate software quality
Development Effort (Hours)	Measure productivity
Schedule Variance	Monitor project timeline
Cost Variance	Track project expenses
Test Coverage	Assess testing effectiveness
Customer Satisfaction	Evaluate user satisfaction

SGOU

Example: A small software company with 12 employees uses a free project management tool to monitor task completion, development effort, and defect count. The tool automatically generates weekly reports, allowing the project manager to track progress, identify delays, and improve project planning without investing in expensive measurement software.

Cost-effective measurement techniques help Very Small Entities (VSEs) measure software projects with less cost and effort by using simple methods, essential metrics, and affordable tools. They help VSEs track performance, improve quality, use resources effectively, and support continuous improvement.

5.2.4.5 Best Practices for Implementing Measurement in VSEs

Implementing software measurement in Very Small Entities (VSEs) requires a practical and simple approach because these organizations often have limited staff, budget, and technical resources. Instead of adopting complex measurement frameworks, VSEs should focus on collecting only the most important project data that supports decision-making and improves software quality. By following best practices, VSEs can implement effective measurement systems with minimal cost and effort while achieving better project performance and continuous improvement.

1. **Define Clear Measurement Objectives:** Identify what needs to be measured and ensure that the measurements support project and business goals.

2. **Select Simple and Relevant Metrics:** Use easy-to-understand metrics such as project completion percentage, defect count, effort, schedule variance, and customer satisfaction.

3. **Measure Only What Is Necessary:** Avoid collecting excessive data. Focus on information that helps improve project performance and decision-making.

4. **Use Standardized Measurement Practices:** Follow recognized standards such as ISO/IEC 29110 to ensure consistency and reliability in measurement activities.

5. **Automate Data Collection Where Possible:** Use simple project management and issue-tracking tools to reduce manual effort and improve data accuracy.

6. **Review Measurement Data Regularly:** Analyze measurement results periodically to monitor project progress, identify issues, and take corrective actions.

7. **Involve the Entire Team:** Encourage developers, testers, and project managers to participate in measurement activities and understand their importance.

8. **Communicate Measurement Results;** Share measurement reports with stakeholders using simple dashboards, charts, or progress reports to support informed decision-making.

SGOU

9. **Continuously Improve the Measurement Process:** Regularly evaluate the effectiveness of the measurement system and update metrics based on project needs and organizational goals.

10. **Keep the Measurement Process Cost-Effective:** Choose affordable tools and techniques that provide useful information without increasing project overhead.

Successful implementation of software measurement in Very Small Entities (VSEs) depends on keeping the measurement process simple, relevant, and cost-effective. By defining clear objectives, selecting meaningful metrics, using standardized practices such as ISO/IEC 29110, and regularly reviewing measurement results, VSEs can improve software quality, manage projects more effectively, reduce risks, and achieve continuous improvement without requiring extensive resources or complex measurement systems.

5.2.5 Comparative Analysis of Software Measurement Models

Software measurement models provide structured approaches for collecting, analyzing, and using measurement data to improve software development and project management. The Practical Software and Systems Measurement (PSM) method focuses on project-level measurement and decision-making, the Capability Maturity Model Integration (CMMI) emphasizes process improvement through systematic measurement, and Measurement in Very Small Entities (VSEs) offers a simplified approach for organizations with limited resources. Although all three models aim to enhance software quality and project performance, they differ in scope, complexity, target organizations, and measurement practices. The following Table 5.2.18 highlights the key differences among these software measurement models.

Table 5.2.18 Comparison of Software Measurement Models

Feature	PSM (Practical Software and Systems Measurement)	CMMI (Capability Maturity Model Integration)	Measurement in VSEs
Primary Focus	Project measurement and management	Process improvement and organizational maturity	Simple measurement for small organizations
Main Objective	Support project planning, monitoring, and decision-making	Improve organizational processes and product quality	Improve project management with minimal resources
Target Organizations	Small, medium, and large organizations	Medium and large organizations	Very Small Entities (up to 25 people)
Measurement Approach	Goal-driven and information-based	Process-oriented and maturity-based	Practical, lightweight, and easy to implement

SGOU

Complexity	Moderate	High	Low
Measurement Scope	Software projects and systems	Organization-wide processes and projects	Small software projects
Key Standard/ Framework	PSM Framework	CMMI Measurement and Analysis (MA) Process Area	ISO/IEC 29110
Data Collection	Project-specific metrics	Standardized organizational metrics	Essential project metrics
Implementation Cost	Moderate	High	Low
Resource Requirement	Moderate	High	Minimal
Suitable For	Organizations seeking effective project control	Organizations aiming for process maturity and continuous improvement	Start-ups and small software development teams
Continuous Improvement	Supported through measurement feedback	Strong emphasis on continuous process improvement	Supported through simple measurement practices

The PSM, CMMI, and VSE measurement models each provide valuable approaches to software measurement, but they serve different organizational needs. PSM is best suited for project-level measurement and decision-making, CMMI provides a comprehensive framework for organizational process improvement, and VSE Measurement offers a lightweight and practical solution for small software organizations. Selecting the appropriate model depends on factors such as organizational size, project complexity, available resources, and long-term business objectives.

SGOU

Recap

- Software measurement models provide structured methods for measuring software products, processes, and projects.
- Measurement models help improve project management, software quality, and decision-making.
- Effective software measurement supports continuous process improvement.
- The Practical Software and Systems Measurement (PSM) method is a project-oriented measurement framework.

- PSM focuses on meeting stakeholder information needs through measurement.
- The PSM framework follows a continuous measurement cycle of planning, collecting, analyzing, and reporting data.
- Information needs are identified before selecting measures in the PSM method.
- Appropriate measures and metrics are selected based on project objectives.
- Measurement data is collected, analyzed, and reported to support project decisions.
- PSM helps identify project risks and monitor project performance.
- The Capability Maturity Model Integration (CMMI) is a framework for process improvement.
- CMMI promotes the use of measurement for effective project and process management.
- The Measurement and Analysis (MA) process area is an important part of the CMMI framework.
- The MA process area focuses on defining, collecting, analyzing, and reporting measurement data.
- Clearly defined measures and metrics improve the accuracy of software measurement.
- Reporting measurement results helps stakeholders make informed decisions.
- CMMI encourages continuous improvement through data-driven management.
- Very Small Entities (VSEs) are organizations with up to 25 people.
- VSEs require simple, practical, and low-cost measurement practices.
- ISO/IEC 29110 provides software engineering guidelines specifically for VSEs.
- Simple metrics such as defects, effort, schedule, and customer satisfaction are commonly used in VSEs.
- Cost-effective measurement techniques reduce the effort and expense of software measurement.

SGOU

- Best practices help VSEs implement measurement successfully with limited resources.
- PSM, CMMI, and VSE measurement models differ in their scope, complexity, and target organizations.
- Selecting the appropriate measurement model depends on organizational size, project requirements, available resources, and improvement goals.

Objective Type Questions

1. What does PSM stand for?
2. Which software measurement model focuses on project management and decision-making?
3. Which framework includes the Measurement and Analysis (MA) process area?
4. What is the full form of CMMI?
5. Which international standard is designed specifically for Very Small Entities (VSEs)?
6. A VSE typically has up to how many people?
7. Which CMMI process area is responsible for software measurement?
8. What is the first step in the PSM measurement process?
9. What type of information is collected during software measurement?
10. Which document provides guidelines for measurement practices in VSEs?
11. What is the primary purpose of software measurement models?
12. Which PSM activity follows the selection of measures?
13. What is a calculated value derived from one or more measures called?
14. What is the raw data collected from a software project called?
15. Which maturity level of CMMI emphasizes quantitative management?
16. Which maturity level of CMMI focuses on continuous improvement?
17. Which CMMI maturity level is known as the "Managed" level?

SGOU

18. Which measurement activity involves examining collected data to identify trends?

19. Which measurement activity communicates results to stakeholders?

20. Which simple metric indicates the percentage of completed work?

21. Which metric measures the number of defects in software?

22. Which metric compares planned and actual completion time?

23. Which technique reduces the cost of measurement by using existing project data?

24. Which type of organization benefits most from ISO/IEC 29110?

25. Which measurement model is most suitable for small software organizations with limited resources?

Answers to Objective Type Questions

1. Practical Software and Systems Measurement
2. PSM
3. CMMI
4. Capability Maturity Model Integration
5. ISO/IEC 29110
6. 25
7. Measurement and Analysis (MA)
8. Planning
9. Data
10. ISO/IEC 29110
11. Improvement
12. Data Collection
13. Metric

SGOU

14. Measure

15. Level 4

16. Level 5

17. Level 2

18. Analysis

19. Reporting

20. Completion Percentage

21. Defect Count

22. Schedule Variance

23. Reuse

24. VSEs

25. VSE Measurement

Assignments

1. Explain the Practical Software and Systems Measurement (PSM) method with its objectives, principles, and measurement framework.

2. Describe the Capability Maturity Model Integration (CMMI) framework and explain the Measurement and Analysis (MA) process area.

3. Explain the concept of Measurement in Very Small Entities (VSEs). Discuss the need, simple metrics, and best measurement practices.

4. Compare the PSM method, CMMI, and Measurement in Very Small Entities (VSEs).

5. Discuss the role of ISO/IEC 29110 in software measurement and explain how it supports measurement practices in Very Small Entities (VSEs).

SGOU

Suggested Reading

1. Software Engineering Institute (SEI) – CMMI Resources https://cmmiinstitute.com/
2. ISO/IEC 29110 – Systems and Software Engineering for Very Small Entities (ISO) https://www.iso.org/standard/51162.html
3. Practical Software Measurement (PSM) Official Website https://www.psmsc.com/
4. IEEE Computer Society – Software Engineering Standards and Resources https://www.computer.org/technical-committees/software-engineering/
5. International Organization for Standardization (ISO) – Software and Systems Engineering Standards https://www.iso.org/committee/45086.html

Reference

1. Chrissis, M. B., Konrad, M., & Shrum, S. (2018). *CMMI for development: Guidelines for process integration and product improvement* (3rd ed.). Addison-Wesley Professional.
2. McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., & Hall, F. (2002). *Practical software measurement: Objective information for decision makers*. Addison-Wesley Professional.
3. Pressman, R. S., & Maxim, B. R. (2024). *Software engineering: A practitioner's approach* (10th ed.). McGraw-Hill Education.
4. ISO/IEC. (2023). *ISO/IEC 29110-4-1: Systems and software engineering- Lifecycle profiles for very small entities (VSEs)- Profile specifications: Generic profile group*. International Organization for Standardization.
5. Sommerville, I. (2020). *Software engineering* (10th ed.). Pearson.

SGOU



Measurement Tools and Risk Management Standards

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the role of measurement tools in software measurement
- describe the purpose, advantages, and challenges of surveys as measurement tools
- identify the major risk management standards and models used in software projects
- analyse how measurement data supports risk identification, assessment, and monitoring

Prerequisite

Software projects involve activities such as requirement analysis, design, coding, testing, and maintenance. During these activities, project teams need information about software quality, development progress, cost, time, and user satisfaction. Understanding how data is collected and interpreted helps learners understand the importance of measurement tools, particularly surveys, in evaluating software products and processes.

Software projects may also face uncertainties such as changing requirements, technical difficulties, resource shortages, and schedule delays. Understanding these situations helps learners recognise the need to identify potential problems, assess their possible effects, and plan appropriate responses. This provides the necessary foundation for studying risk management standards such as ISO/IEC/IEEE 16085 and ISO 31000, and risk management practices associated with CMMI.

Keywords

Software Measurement, Survey, Risk Management, ISO Standards, CMMI Model, Measurement Tools

SGOU

Discussion

5.3.1 Introduction to Measurement Tools

Software measurement involves the systematic collection, analysis, and interpretation of quantitative information related to software products, processes, and projects. Measurement helps organisations understand the quality, progress, cost, effort, productivity, and performance of software development activities. Measurement tools support this process by helping collect, organise, analyse, and present measurement data in a systematic manner.

Measurement tools are useful because they provide objective information for decision-making. They help project teams monitor software development activities, identify deviations from planned objectives, detect possible problems, and support continuous improvement. Thus, measurement tools provide evidence that can be used for evaluating and improving software projects.

5.3.2 The Survey as a Measurement Tool

A survey is a systematic method of collecting information, opinions, perceptions, and experiences from a selected group of people. In software engineering, surveys can be used to collect information from developers, project managers, customers, users, testers, and other stakeholders. A survey is particularly useful when information cannot be obtained directly through technical measurements.

For example, software usability, user satisfaction, employee perceptions, and stakeholder expectations may not be completely measured through technical metrics. A survey can collect such information using a structured set of questions. The responses can then be analysed to understand the current situation and identify areas for improvement.

Purpose and Importance of Surveys

The main purpose of a survey is to collect structured information from people who are associated with a software product or process. Surveys help organisations understand user satisfaction, development problems, stakeholder expectations, process effectiveness, and perceptions about software quality.

Surveys are also useful for collecting information from a large number of participants within a relatively short period. The results can support decision-making and continuous improvement. For example, users may be asked to rate the usability of an application on a five-point scale. The collected responses can be analysed to determine the overall level of user satisfaction.

Quantitative and Qualitative Data in Surveys

Surveys can collect both quantitative and qualitative information. Quantitative data are expressed using numbers, ratings, frequencies, percentages, or predefined categories. For example, users may rate the usability of software from 1 to 5.

SGOU

Qualitative data are obtained through open-ended questions that allow participants to describe their experiences, difficulties, and suggestions. For example, users may be asked to describe the problems they faced while using a software application.

Table 5.3.1 Comparison of Quantitative and Qualitative data

Type of Data	Survey Method	Example
Quantitative	Rating scale	Rate software usability from 1 to 5
Quantitative	Frequency-based question	Number of errors experienced
Qualitative	Open-ended question	Describe difficulties faced while using the software
Qualitative	Comments and suggestions	Suggest improvements to the software

Advantages of Surveys

Surveys provide a simple and systematic method for collecting information from different groups of stakeholders. They can collect data from a large number of people at a relatively low cost. Online surveys can further reduce the time and effort required for data collection.

Surveys also help organisations obtain different perspectives about the same software product or process. The results can be used to identify problems, understand stakeholder expectations, and support continuous improvement.

Challenges of Survey-Based Measurement

Survey results may be affected by response bias because participants may not always provide completely accurate opinions. A low participation rate can also reduce the reliability of the collected information. Since many survey responses are based on personal opinions, subjectivity is another important challenge.

The quality of a survey also depends on proper question design. Unclear, confusing, or leading questions may produce unreliable results. Therefore, survey questions should be carefully designed according to the purpose of the measurement.

5.3.3 Introduction to Software Risk Management

Software projects involve uncertainty, and unexpected events can affect project cost, schedule, quality, security, and performance. Software risk management is a systematic approach used to identify, analyse, treat, and monitor risks that may affect a software project.

A software risk is an uncertain event or condition that may negatively affect project objectives if it occurs. For example, if a project depends on a technology with which the development team has limited experience, lack of technical knowledge may become a risk.

Risk management helps project teams identify possible problems early and prepare suitable responses. It supports better planning, resource allocation, decision-making, and project control.

SGOU

The major activities of risk management are risk identification, risk analysis, risk treatment, and risk monitoring. These activities are performed continuously because existing risks may change and new risks may appear during the software life cycle.

5.3.3.1 Risk Management According to Standards

Standards provide systematic and consistent approaches for managing risks. They provide guidance on identifying, assessing, treating, and monitoring risks throughout the software and organisational life cycle.

ISO/IEC/IEEE 16085 – Risk Management Process

ISO/IEC/IEEE 16085 provides guidance for managing risks throughout the life cycle of systems and software projects. It provides a structured approach for integrating risk-management activities into software and system life-cycle processes.

The main activities include establishing the risk-management approach, identifying risks, analysing and evaluating risks, treating risks, monitoring and communicating risks, and maintaining risk information. These activities help organisations manage risks systematically throughout the project life cycle.

For example, a software project may identify a risk related to the unavailability of skilled developers. The project team can analyse its possible effect, plan an appropriate response, and continuously monitor the situation.

ISO 31000 – Risk Management Guidelines

ISO 31000 provides principles, a framework, and a process for managing risks in organisations. It can be applied to different types of organizations and different types of risks. It helps integrate risk management with planning, decision-making, and organisational activities.

ISO 31000 emphasises that risk management should be integrated into organisational activities, structured and comprehensive, customised to the organisation's needs, inclusive of relevant stakeholders, dynamic, based on appropriate information, and continuously improved.

The framework supports activities such as leadership and commitment, integration, implementation, evaluation, and improvement. Thus, ISO 31000 provides a general framework for managing risk systematically.

5.3.3.2 Risk Management According to Models

CMMI Risk Management Model

CMMI provides practices for improving organisational and project performance. Risk management in CMMI focuses on identifying potential problems, analysing their possible effects, planning appropriate responses, and monitoring risks continuously.

Risk identification involves finding and documenting possible events that may affect project objectives. Risk analysis evaluates the likelihood and possible impact of

SGOU

identified risks. Risk handling involves selecting suitable actions to deal with risks, such as avoiding, reducing, transferring, or accepting them. Risk monitoring involves regularly reviewing risks and checking whether the planned responses are effective.

CMMI therefore treats risk management as a continuous activity rather than an activity performed only at the beginning of a project.

5.3.3.3 Risk Management Models and Techniques

Risk management techniques help project teams organise and assess risks systematically.

- **Risk Breakdown Structure:** A Risk Breakdown Structure (RBS) is a hierarchical method of organising risks into categories. It provides a structured view of possible sources of risk in a software project. For example, risks may be grouped according to technical, project, organisational, resource, and external sources.

- **Probability–Impact Analysis:** Probability–Impact Analysis evaluates a risk according to two important factors: the probability that the risk will occur and the impact it may have on the project. A risk with high probability and high impact normally requires greater attention than a risk with low probability and low impact.

- **Risk Matrix:** A Risk Matrix is a visual method for combining probability and impact to determine the level of a risk. It helps project teams compare risks and prioritise those that require immediate attention. Risk levels may generally be represented as low, medium, high, or very high.

- **Failure Mode and Effects Analysis:** Failure Mode and Effects Analysis (FMEA) is a systematic technique for identifying possible failures and examining their effects. In software projects, it can be used to identify possible failures in software functions, processes, or components and determine which failures require attention.

5.3.3.4 Relationship Between Measurement and Risk Management

Measurement and risk management are closely connected. Measurement provides objective information that can help project teams identify and monitor risks. For example, an increasing number of software defects may indicate a quality risk, while increasing development time may indicate a schedule risk.

Measurement data can also help teams monitor changes in risk levels and evaluate whether risk responses are effective. Metrics such as defect counts, development effort, project cost, schedule progress, and resource usage can provide early indications of potential problems.

Therefore, combining measurement with risk management helps software teams identify risks early, make informed decisions, and take appropriate corrective or preventive actions.

SGOU

5.3.3.5 Benefits and Challenges of Risk Management Standards

Risk management standards provide a consistent and systematic approach to managing uncertainty. They help organisations improve project planning, identify problems early, allocate resources effectively, and improve software quality and reliability.

However, risk management also has challenges. It may be difficult to predict every possible risk, especially when historical information is limited. Changing requirements and project conditions may introduce new risks. Effective risk management also requires continuous monitoring and active participation from the project team.

Measurement tools help software organisations collect and analyse information about software products, processes, and projects. The survey is an important measurement tool for collecting quantitative and qualitative information from users, developers, customers, and other stakeholders. Surveys are useful for measuring aspects such as user satisfaction, perceptions, expectations, and software usability.

Risk management provides a systematic approach for dealing with uncertainty in software projects. Standards such as ISO/IEC/IEEE 16085 and ISO 31000 provide structured guidance for managing risks, while models such as CMMI provide practices for identifying, analysing, treating, and monitoring risks. Techniques such as RBS, Probability–Impact Analysis, Risk Matrix, and FMEA support risk assessment. The combination of measurement and risk management helps organisations identify problems early, make evidence-based decisions, and improve software project outcomes.

Recap

- Measurement tools help collect and analyse information about software products, processes, and user experiences.
- A survey is a measurement tool used to collect opinions, feedback, and information from users and stakeholders.
- Surveys collect quantitative data in numerical form and qualitative data in descriptive form.
- Surveys help measure software usability, user satisfaction, and stakeholder expectations.
- Response bias, unclear questions, and low response rates are common challenges in conducting surveys.
- Software risk management is the systematic process of identifying, analysing, treating, and monitoring risks in a software project.
- ISO/IEC/IEEE 16085 provides guidance on risk management in systems and software life-cycle processes.
- ISO 31000 provides general principles and guidelines for risk management.

SGOU

- CMMI supports process improvement and includes practices related to managing project risks.
- The Risk Breakdown Structure (RBS) organises risks into categories, while Probability–Impact Analysis evaluates the likelihood and impact of risks.
- A Risk Matrix helps classify and prioritise risks based on their probability and impact.
- Failure Mode and Effects Analysis (FMEA) identifies possible failures and examines their effects.
- Measurement data supports risk identification, assessment, monitoring, and decision-making in software projects.
- Effective risk management helps reduce uncertainty, prevent potential problems, and improve the chances of project success.

Objective Type Questions

1. What tool collects opinions and feedback from software users?
2. Which type of survey data is expressed numerically?
3. Which type of survey data describes experiences in words?
4. What bias occurs when survey participants provide inaccurate or socially acceptable responses?
5. What is the systematic management of software uncertainties called?
6. Which ISO standard provides guidance for systems and software life-cycle risk management?
7. Which ISO standard provides general risk management guidelines?
8. Which model provides practices for improving organisational and project performance?
9. Which structure organises risks into categories hierarchically?
10. Which analysis evaluates the likelihood and impact of a risk?
11. Which visual tool combines risk probability and impact?
12. What is the abbreviation for Failure Mode and Effects Analysis?

SGOU

13. Which risk management activity identifies possible problems?

14. Which risk management activity evaluates the likelihood and impact of risks?

15. Which risk management activity tracks risks continuously?

Answers to Objective Type Questions

1. Survey.
2. Quantitative.
3. Qualitative.
4. Response bias.
5. Risk management.
6. ISO/IEC/IEEE 16085.
7. ISO 31000.
8. CMMI.
9. RBS.
10. Probability–Impact Analysis.
11. Risk Matrix.
12. FMEA.
13. Identification.
14. Analysis.
15. Monitoring.

SGOU

Assignments

1. Explain the concept and importance of measurement tools in software engineering. Discuss the survey as a measurement tool, including its purpose, advantages, and challenges.

2. Describe how surveys collect quantitative and qualitative data. Explain their role in measuring software usability, user satisfaction, and stakeholder expectations with suitable examples.

3. Explain the principles and major activities of risk management according to ISO/IEC/IEEE 16085 and ISO 31000.

4. Discuss risk management according to the CMMI model. Explain risk identification, risk analysis, risk handling, and risk monitoring with suitable examples.

5. Explain the Risk Breakdown Structure, Probability–Impact Analysis, Risk Matrix, and Failure Mode and Effects Analysis as risk management techniques.

6. Analyse the relationship between measurement and risk management in software projects. Explain how measurement data supports risk identification, monitoring, and decision-making.

Suggested Reading

1. International Organization for Standardization (ISO) – ISO 31000 Risk Management Guidelines ISO 31000 – Risk Management

2. International Organization for Standardization (ISO) – ISO/IEC/IEEE 16085 Risk Management ISO/IEC/IEEE 16085 – Risk Management

3. Project Management Institute (PMI) – Risk Management Resources PMI Risk Management

4. Carnegie Mellon Software Engineering Institute (SEI) – Software Measurement Resources SEI – Software Measurement

5. Project Management Institute (PMI) – Risk Management Practice and Standards PMI – Risk Management Standard

Reference

1. O'Regan, G. (2025). *Guide to software project management*. Springer.

2. Radaideh, M. A. (2023). *Software project management*. De Gruyter.

3. Suvvari, S. K., & Jarubula, R. (2024). *Project management and software engineering*. Academic Guru Publishing House.

4. Chandramouli, S., & Dutt, S. (n.d.). *Software project management*. Pearson Education India.

5. Ebert, C., & Dumke, R. (2007). *Software measurement: Establish – extract – evaluate – execute*. Springer.

SGOU



Practical Considerations for Risk Management

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the purpose and methods of risk evaluation in software projects
- describe the major strategies used for risk control and contingency planning
- identify the responsibilities of different participants in risk management
- explain the planning, execution, and monitoring activities of Software Quality Assurance

Prerequisite

In a software development project, a team may face problems such as delays in completing tasks, unexpected expenses, software defects, security vulnerabilities, and failure to meet user requirements. For example, if a software project depends on a third-party service and that service becomes unavailable, the project may experience delays. Identifying such situations in advance and understanding their possible consequences help the team prepare suitable responses and reduce their effects on the project.

Effective risk management requires knowledge of project planning, software development activities, quality assurance, and problem-solving methods. Understanding how to identify potential risks, estimate their probability and impact, select appropriate control measures, and assign responsibilities helps ensure successful project completion. Knowledge of Software Quality Assurance (SQA) is also essential because it helps establish quality standards, plan quality-related activities, monitor their execution, and improve the reliability of the final software product. These concepts provide the foundation for studying risk evaluation, risk control, risk management roles, and SQA planning and execution.

Keywords

Risk Evaluation, Risk Control, Risk Management Roles, SQA Planning, Quality Assurance

SGOU

Discussion

5.4.1 Overview

Software projects may face various uncertainties during development. These uncertainties can affect project cost, schedule, software quality, security, performance, and customer satisfaction. For example, a project may experience delays because of changing requirements, a shortage of skilled developers, technical failures, or problems with an external service. Practical risk management helps a project team identify such possibilities early and take suitable actions to reduce their effects.

Risk evaluation helps the team understand the likelihood and potential consequences of risks. Risk control involves selecting and implementing appropriate responses. Risk management roles establish responsibility for identifying, assessing, and handling risks. Software Quality Assurance (SQA) planning and execution support these activities by establishing quality procedures and checking whether they are followed. Together, these practices help a team reduce uncertainty, prevent avoidable problems, and deliver reliable software.

5.4.2 Risk Evaluation

Risk evaluation is the process of examining identified risks to determine their likelihood, potential impact, and significance to a software project. It helps the team decide which risks require immediate attention and which can be monitored periodically. For example, if a project depends on one developer who has specialised technical knowledge, the team may evaluate the possibility of that developer leaving and the resulting effect on the project schedule.

Risk evaluation supports informed decisions about resource allocation, project planning, and risk response. It does not necessarily eliminate risks; instead, it helps the team understand their importance and decide how to manage them.

5.4.2.1 Risk Identification and Evaluation

Risk identification is the process of recognising events or conditions that may negatively affect a project. Risk evaluation follows identification and determines the likelihood and consequences of each risk.

Software project risks commonly arise from several sources. Technical risks include technology failures, integration difficulties, and performance problems. Project risks include schedule delays, budget overruns, and resource shortages. People-related risks include staff turnover, inadequate skills, and communication difficulties. Requirement risks arise from unclear or frequently changing requirements. Security risks include unauthorised access and data breaches, while external risks may result from vendor failures, regulatory changes, or service interruptions.

For example, when a project uses an external cloud service, the team may identify service interruption as a risk. It then evaluates how likely the interruption is and how seriously it could affect application availability.

SGOU

5.4.2.2 Risk Probability and Impact

Two important factors in risk evaluation are probability and impact. Risk probability refers to the likelihood that a particular risk will occur. It may be classified as very low, low, medium, high, or very high. Risk impact refers to the consequences if the risk occurs. These consequences may affect project cost, schedule, software quality, security, system performance, or customer satisfaction.

A simple risk score can be calculated using the following formula:

Risk Score = Probability x Impact

Suppose a risk receives a probability rating of 4 and an impact rating of 5 on a five-point scale.

$$Risk\ Score = 4 \times 5 = 20$$

The resulting score of 20 indicates that the risk deserves significant attention under the project's rating scheme. The organisation should define the rating scale and score ranges it uses; a score should not be interpreted without considering that scheme.

5.4.2.3 Risk Exposure and Risk Rating

Risk exposure estimates the potential loss associated with a risk. When probability is expressed as a numerical value between 0 and 1 and the potential loss is expressed in monetary terms, the formula is:

Risk Exposure =Probability of Occurrence x Potential Loss

For example, suppose a software project has a 20% probability of a failure that would cost ₹5,00,000 to resolve.

$$RiskExposure = 0.20 \times ₹5{,}00{,}000 = ₹1{,}00{,}000$$

The estimated risk exposure is ₹1,00,000. This is an expected-loss estimate, not a guarantee that the project will lose that exact amount.

Risk rating classifies risks according to their assessed probability and impact. A project may use low, medium, and high categories to determine the action required. For example, low risks may be monitored, medium risks may require a mitigation plan, and high risks may need immediate action. The precise score ranges should be established by the organisation or project team.

5.4.2.4 Qualitative Risk Evaluation

Qualitative risk evaluation assesses risks using descriptive categories instead of relying mainly on numerical estimates. It is useful when detailed statistical data are unavailable or when the team needs to evaluate risks quickly during early project planning.

A common technique is the Probability–Impact Matrix, which combines the likelihood of a risk with its potential consequences. Risks with high probability and high impact

SGOU

are usually given greater priority than risks with low probability and low impact.

For example, frequent changes to customer requirements may be rated high in probability and high in impact. The team may therefore prioritise requirement clarification, stakeholder communication, and change-control procedures.

The main advantages of qualitative evaluation are that it is easy to understand, requires relatively little data, and supports rapid prioritisation. However, its results may depend on the judgement and experience of the people assessing the risks.

5.4.2.5 Quantitative Risk Evaluation

Quantitative risk evaluation uses numerical data and mathematical techniques to estimate the effects of risks. It is particularly useful when reliable historical data or financial estimates are available. Techniques include probability calculations, Expected Monetary Value (EMV), decision tree analysis, sensitivity analysis, statistical analysis, and simulation.

Expected Monetary Value is calculated as:

EMV = Probability x Financial Impact

Suppose a software project has a 10% probability of experiencing a failure that would cause a financial loss of ₹2,00,000.

$$EMV = 0.10 \times ₹2,00,000 = ₹20,000$$

The expected monetary value is ₹20,000. This estimate can help the team compare risk-response options and plan resources.

Quantitative evaluation supports financial planning and comparisons between alternatives. However, it requires reasonably reliable data, and its estimates may be inaccurate when probabilities or potential losses are uncertain.

5.4.2.6 Risk Prioritisation

Risk prioritisation is the process of ranking risks according to their significance so that limited resources can be directed towards the most serious problems. The team may consider probability, impact, risk exposure, urgency, and the availability of effective response options.

Consider the following example:

Table 5.4.1 Priority of different Risks

Risk	Probability	Impact	Suggested priority
Server failure	High	High	High
Minor user-interface defect	High	Low	Medium
Change in reporting format	Low	Medium	Low

SGOU

Server failure receives the highest priority because it could seriously affect system availability. The minor user-interface defect may require correction but is less likely to disrupt the entire project. The reporting-format change can be monitored and handled according to its actual effect.

Risk evaluation is therefore an essential part of project decision-making. It helps the team understand risks, compare their significance, and select suitable control measures.

5.4.3 Risk Control

Risk control is the process of taking planned actions to avoid, reduce, transfer, or accept risks that may affect a software project. It follows risk evaluation and focuses on managing risks according to their importance. The objective is not always to eliminate every risk; in many situations, the practical goal is to reduce its probability or impact to an acceptable level.

For example, if a project depends on a single developer who understands an important software module, the team can reduce the risk by documenting the code, sharing technical knowledge, and training another developer. These measures make the project less dependent on one individual.

5.4.3.1 Risk Mitigation Strategies

Risk mitigation refers to actions taken to reduce the likelihood of a risk or minimise its consequences. The appropriate strategy depends on the nature of the risk, its severity, and the resources available to manage it.

The main risk-response strategies include risk avoidance, risk reduction, risk transfer, and risk acceptance. Contingency planning is also important because it prepares the team to respond if a risk actually occurs. For example, a project expecting performance problems under heavy user traffic may conduct load testing, optimise database queries, and prepare additional server capacity.

5.4.3.2 Risk Avoidance

Risk avoidance involves changing a project plan, technology, process, or requirement to eliminate a particular risk. It is appropriate when the potential consequences are serious and a practical alternative is available.

For example, a team may choose a well-established software framework instead of an experimental framework that has limited support and uncertain reliability. Similarly, a project may remove an unnecessary high-risk feature or replace an unreliable third-party service with a more dependable alternative.

Although avoidance can remove a particular source of risk, it may introduce other costs or limitations. Therefore, the alternative should be evaluated before the decision is made.

SGOU

5.4.3.3 Risk Reduction

Risk reduction aims to decrease the probability of a risk occurring or reduce its impact if it occurs. It is often more practical than complete avoidance because many software risks cannot be eliminated entirely.

Common risk-reduction measures include regular testing, code reviews, staff training, data backups, security controls, performance monitoring, prototypes, and proper documentation. For example, the risk of losing important project data can be reduced through regular backups, verification of backup files, and testing of recovery procedures.

The effectiveness of risk reduction should be reviewed regularly. If a control measure does not reduce the risk sufficiently, the team may need to introduce additional measures.

5.4.3.4 Risk Transfer and Risk Acceptance

Risk transfer involves transferring some responsibility or financial consequences of a risk to another party. This may be achieved through insurance, contractual arrangements, outsourcing, or the use of managed services. For example, an organisation may use a managed cloud service to transfer certain infrastructure-management responsibilities to the provider. However, transferring responsibility does not necessarily remove all risk; the organisation may still be accountable for data protection, access control, and compliance.

Risk acceptance means consciously deciding to tolerate a risk when further action is unnecessary, impractical, or more expensive than the likely consequences. For example, a team may accept a minor delay in a non-critical reporting feature if the cost of preventing the delay is disproportionate to its impact.

Risk acceptance can be active or passive. Active acceptance may include preparing a contingency plan or reserving resources, while passive acceptance involves monitoring the risk without making additional preparations. Acceptance should be a deliberate decision rather than the result of ignoring a known risk.

5.4.3.5 Contingency Planning

Contingency planning involves preparing alternative actions to be followed if a risk occurs. It helps the team respond quickly and minimise disruption. A contingency plan generally identifies the possible event, the conditions that trigger a response, the actions to be taken, the responsible people, and the resources required.

For example, if the main project server becomes unavailable, the contingency plan may specify switching to a backup server, restoring verified data, and informing the relevant stakeholders.

A contingency plan should be reviewed and updated as the project changes. Team members should understand their responsibilities so that the response can be implemented without unnecessary delay.

SGOU

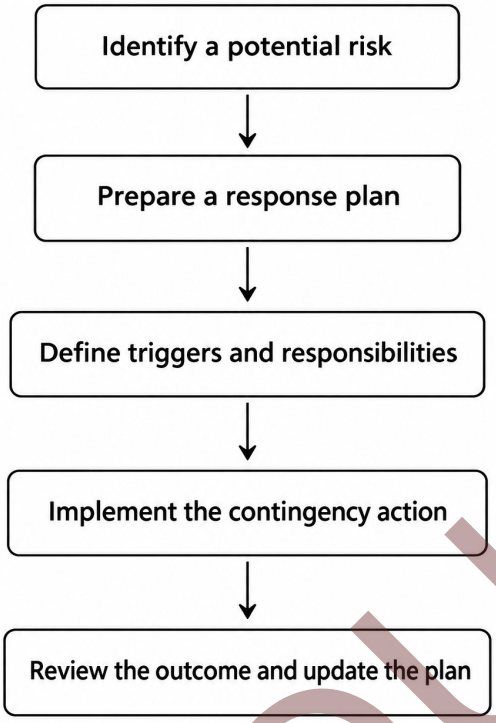


Fig. 5.4.1 Contingency Planning Process

5.4.3.6 Risk Monitoring and Control

Risk monitoring and control is the continuous process of tracking known risks, checking whether response measures are effective, and identifying new risks. A risk that was initially considered minor may become serious when project conditions change.

The team should review the risk register, monitor warning indicators, update probability and impact assessments, track mitigation activities, and communicate significant changes. Risks that are no longer relevant can be closed after appropriate review.

For example, if software testing reveals a growing number of critical defects, the team may increase the risk rating for the affected module and allocate additional testing resources. After corrective action, the team should reassess whether the risk has been reduced.

Effective risk control is therefore a continuous activity throughout the software development life cycle. It helps protect the project schedule, budget, quality, and resources.

5.4.4 Risk Management Roles

Risk management requires cooperation among different people involved in a software project. Each participant has specific responsibilities for identifying, evaluating, responding to, or monitoring risks. Clearly defined roles improve accountability, communication, and decision-making.

If responsibilities are not assigned properly, important risks may remain unnoticed, or responses may be delayed. For example, assigning a team member to monitor cybersecurity risks helps ensure that security problems are reported promptly and handled by the appropriate people.

5.4.4.1 Project Manager's Role

The project manager coordinates risk management activities and ensures that risks are considered during project planning and execution. The project manager oversees the risk management approach, assigns risk owners, reviews risk status, allocates resources, and communicates significant risks to stakeholders.

For example, if a project falls behind schedule because of a shortage of developers, the project manager may assess the situation, reallocate resources, revise the schedule, and arrange appropriate corrective action.

5.4.4.2 Risk Manager or Coordinator's Role

A risk manager or coordinator supports the systematic recording, assessment, and monitoring of risks. In a large project, this may be a dedicated role; in a smaller project, these responsibilities may be assigned to an existing team member.

The coordinator helps maintain the risk register, organise risk review meetings, track response actions, prepare risk reports, and communicate changes in risk status. The role supports the project manager by ensuring that risk information remains organised and up to date.

5.4.4.3 Development Team's Role

The development team identifies and manages risks associated with software design, architecture, coding, integration, technology, and performance. Developers are often well placed to recognise technical problems because they work directly with the software.

For example, developers may discover that an outdated third-party library creates security or compatibility risks. They can report the problem, evaluate alternatives, and implement an appropriate solution.

5.4.4.4 Quality Assurance Team's Role

The Quality Assurance (QA) team identifies and manages risks related to software quality, reliability, performance, security, and compliance. Its responsibilities include planning quality activities, reviewing software work products, conducting tests and audits, tracking defects, and verifying corrective actions.

For example, if testing reveals repeated critical defects in a payment module, the QA team reports the issue and verifies that the defects are corrected before the software is released.

5.4.4.5 Stakeholders' Role

Stakeholders include customers, users, sponsors, management, vendors, and other parties affected by the project. They provide information about business requirements,

SGOU

priorities, expectations, and external conditions that may create risks.

For example, a customer may inform the team about a forthcoming change in business requirements. Early communication allows the team to assess the consequences for project scope, cost, schedule, and quality.

5.4.4.6 Communication and Collaboration

Effective risk management depends on timely communication among all participants. Team members should report emerging risks, share relevant information, clarify ownership, and discuss whether existing response plans remain suitable. Regular risk-review meetings and an up-to-date risk register can support this process.

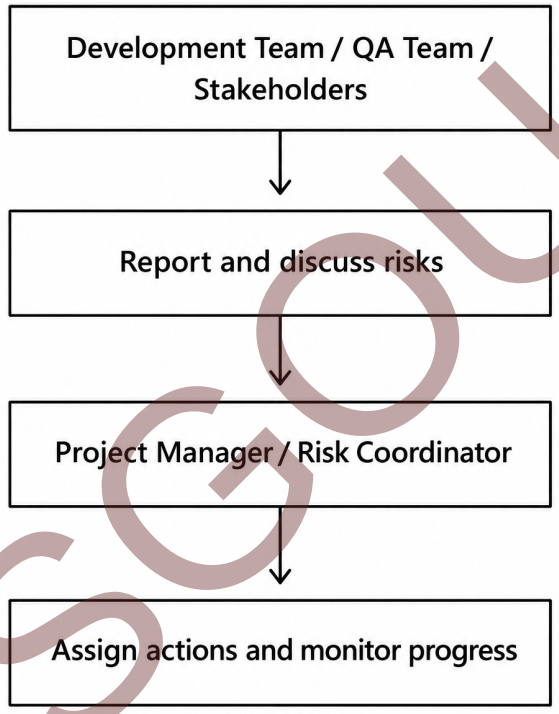


Fig. 5.4.2 Risk Communication and Coordination

Risk management is a shared responsibility. The project manager and risk coordinator help organise the process, while developers, QA personnel, and stakeholders contribute their knowledge and carry out assigned actions.

5.4.5 SQA Planning and Execution

Software Quality Assurance (SQA) is a systematic approach to ensuring that software development processes and products meet specified requirements, established standards, and customer expectations. SQA focuses on both the quality of the development process and the quality of the resulting software.

SQA planning determines the quality objectives, standards, procedures, responsibilities, and activities required for a project. SQA execution puts the approved plan into practice and checks whether quality activities are performed as intended. For example, before releasing an online banking application, the SQA team may plan security testing, code reviews, performance testing, and defect tracking, and then verify that these activities have been completed and critical defects have been addressed.

5.4.5.1 Objectives of SQA Planning

SQA planning establishes a clear approach to achieving and maintaining software quality. It defines the standards to be followed, the quality activities to be performed, the responsibilities of team members, and the methods used to measure and monitor quality.

The main objectives are to prevent defects, identify problems early, ensure compliance with requirements, establish consistent development procedures, and improve customer satisfaction. SQA planning also helps the team determine the tools, resources, documentation, and corrective actions needed to maintain quality throughout development.

5.4.5.2 Components of an SQA Plan

An SQA plan is a document describing how quality assurance will be carried out in a software project. Its contents depend on the project's size, complexity, and requirements, but a typical plan includes the following components.

- **Purpose and scope:** Defines the boundaries and objectives of quality assurance.
- **Quality objectives and standards:** Specifies expected quality levels and applicable standards.
- **Roles and responsibilities:** Identifies the people responsible for reviews, testing, audits, and other activities.
- **SQA activities and procedures:** Describes planned reviews, inspections, testing, and audits.
- **Quality metrics:** Identifies measurements used to evaluate quality, such as defect density and test coverage.
- **Defect management:** Explains how defects are reported, classified, assigned, corrected, and verified.
- **Tools and documentation:** Identifies the tools, records, and reports required.
- **Reporting and corrective actions:** Defines how quality results are communicated and how problems are addressed.

For example, an SQA plan for a web application may require peer review of major code changes, security testing before release, documented defect reports, and approval of the final test results.

SGOU

5.4.5.3 SQA Standards and Procedures

SQA standards and procedures provide consistency in software development and quality assurance. A standard defines an expected requirement or set of guidelines, whereas a procedure describes how a particular activity should be performed.

Depending on the project, organisations may use standards and guidelines such as ISO 9001 for quality management systems, ISO/IEC/IEEE 12207 for software life-cycle processes, and ISO/IEC/IEEE 29119 for software testing. The specific standards selected should be appropriate to the project's needs.

For example, a development team may establish a coding procedure requiring every major code change to undergo peer review before it is merged into the main codebase. Following documented procedures makes quality activities more consistent and easier to verify.

5.4.5.4 SQA Activities and Tasks

SQA activities are performed throughout the software development life cycle to verify that quality requirements are addressed. They include reviewing requirements and designs, inspecting code, planning and conducting tests, checking documentation, monitoring quality measurements, and verifying compliance with established procedures.

These activities help identify defects before they become expensive or difficult to correct. For example, a requirements review may reveal an unclear business rule before developers implement it, preventing rework later in the project.

SQA also supports configuration and change management by checking that software changes are properly recorded, reviewed, and tested. The activities selected should reflect the risks and quality requirements of the project.

5.4.5.5 SQA Execution and Monitoring

SQA execution involves carrying out the activities defined in the approved SQA plan. Monitoring checks whether these activities are completed, whether quality objectives are being met, and whether corrective actions are effective.

The QA team may monitor measures such as the number of defects, defect density, test coverage, test pass rate, defect resolution time, and customer-reported defects. These measurements help the team understand the current quality status and identify areas requiring improvement.

For example, if the number of critical defects increases significantly during system testing, the team may investigate the causes, conduct additional reviews, and improve test coverage. The results should be documented and communicated to the relevant stakeholders.

5.4.5.6 Reviews, Audits, and Inspections

Reviews, audits, and inspections are important SQA techniques, but they serve different purposes.

SGOU

Reviews involve examining software work products, such as requirements, designs, code, or test plans, to identify problems and improve quality. A code review, for example, may identify unclear logic or a possible performance problem.

Audits examine whether project activities and products comply with established standards, procedures, and approved plans. An SQA audit may check whether the project follows its configuration management procedure.

Inspections are structured examinations of work products intended to identify defects and deviations from requirements or standards. A formal code inspection may reveal incorrect logic, missing checks, or violations of coding standards.

These techniques complement software testing. Reviews and inspections can identify problems in work products, while audits provide evidence of compliance with defined processes.

5.4.5.7 Defect Tracking and Corrective Actions

Defect tracking is the process of recording, monitoring, resolving, and verifying defects discovered during development and testing. A structured defect management process helps ensure that reported problems are not overlooked.

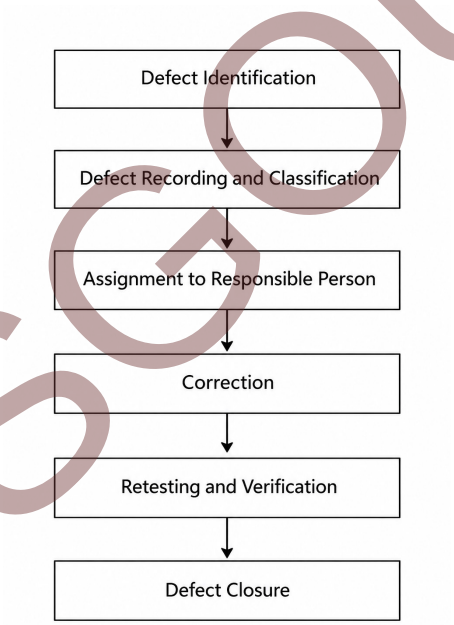


Fig 5.4.3 Defect Tracking Process

Defects may be classified by severity, priority, type, source, and status. Severity indicates the seriousness of the defect, while priority indicates how urgently it should be addressed.

Corrective actions address identified problems and, where possible, their underlying causes. For example, if repeated defects result from unclear requirements, the organisation may introduce more detailed requirements reviews and obtain stakeholder confirmation before development begins.

SGOU

5.4.5.8 SQA Reporting and Documentation

SQA reporting communicates the quality status of a project to the people responsible for making decisions. Reports may describe completed testing, identified defects, audit findings, compliance issues, unresolved critical problems, and corrective actions. Common SQA records include the SQA plan, review reports, audit reports, inspection records, test reports, defect reports, quality metrics, and corrective-action records. These documents provide evidence of the activities performed and help teams trace decisions and follow up on unresolved issues.

For example, a weekly SQA report may show the number of tests completed, the number of defects remaining, the status of critical issues, and the actions required before release. Reports should be accurate, clear, consistent, and kept up to date.

5.4.5.9 Continuous Quality Improvement

Continuous quality improvement is the ongoing effort to improve software products, development processes, and quality assurance practices using measurements, feedback, and lessons learned. Rather than treating quality as a one-time activity, organisations regularly review results and make improvements.

The improvement process can be represented as follows:

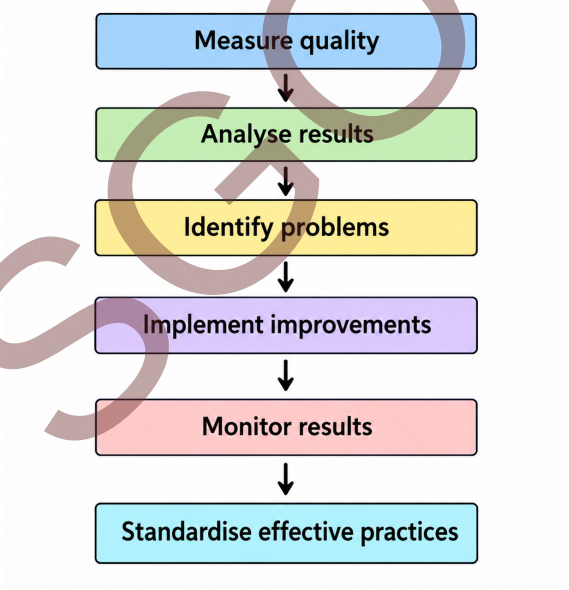


Fig 5.4.4 Continuous Quality Improvement Process

For example, if project data show that many defects originate during requirements development, the organisation may strengthen requirements reviews, improve communication with stakeholders, and provide additional training. It can then measure whether these changes reduce defects in later projects.

Continuous improvement helps reduce recurring problems, increase efficiency, and improve the reliability of software products.

SGOU

Practical considerations for risk management include risk evaluation, risk control, risk management roles, and SQA planning and execution. Risk evaluation determines the probability, impact, exposure, and priority of risks. Risk control uses appropriate strategies such as avoidance, reduction, transfer, acceptance, and contingency planning to manage them. Clearly defined roles and effective communication ensure that risks are identified and addressed promptly. SQA planning establishes quality objectives, standards, responsibilities, and procedures, while SQA execution applies these plans through reviews, inspections, testing, defect tracking, audits, and reporting. Continuous quality improvement uses measurements and lessons learned to strengthen software processes and products. Together, these practices help reduce uncertainty, control project risks, and deliver reliable software.

Recap

- Risk evaluation determines the probability, impact, and significance of risks that may affect a software project.
- Risk probability indicates the likelihood of a risk occurring, while risk impact describes its possible consequences.
- Risk exposure estimates the expected loss associated with a risk, and risk scoring helps assess its priority.
- Qualitative risk evaluation uses descriptive ratings, whereas quantitative risk evaluation uses numerical data and calculations.
- Risk prioritisation helps the project team focus on risks that require immediate attention.
- Risk control involves selecting and implementing suitable actions to manage identified risks.
- Risk avoidance, reduction, transfer, and acceptance are the major strategies used to respond to risks.
- Contingency planning prepares alternative actions to minimise disruption when a risk occurs.
- Risk monitoring and control involves tracking risks, reviewing response measures, and identifying new risks throughout the project.
- Risk management roles define the responsibilities of the project manager, risk coordinator, development team, Quality Assurance (QA) team, and stakeholders.
- Software Quality Assurance (SQA) planning establishes quality objectives, standards, procedures, responsibilities, and activities for a software project.

SGOU

- SQA execution and monitoring ensure that planned quality activities are carried out and quality objectives are evaluated.
- Reviews, audits, and inspections help identify defects, verify compliance, and improve the quality of software work products and processes.
- Defect tracking and corrective actions ensure that identified problems are recorded, resolved, and verified.
- Continuous quality improvement uses quality measurements, feedback, and lessons learned to improve software processes and product reliability.

Objective Type Questions

1. What process determines the probability and impact of identified risks?
2. What factor indicates the likelihood of a risk occurring?
3. What factor describes the consequences of a risk?
4. What technique combines risk probability and impact in a matrix?
5. What is the process of ranking risks according to their significance?
6. Which risk-response strategy eliminates a particular source of risk?
7. Which strategy reduces the probability or impact of a risk?
8. Which strategy transfers some risk responsibility to another party?
9. What plan prepares alternative actions when a risk occurs?
10. Who coordinates the overall management of project risks?
11. Which team identifies defects through testing and reviews?
12. What document describes the quality assurance approach for a project?
13. What activity checks compliance with established procedures and standards?
14. What process records and monitors software defects?
15. What ongoing effort improves software processes and products?

SGOU

Answers to Objective Type Questions

1. Risk evaluation.
2. Probability.
3. Impact.
4. Probability–Impact Matrix.
5. Risk prioritisation.
6. Risk avoidance.
7. Risk reduction.
8. Risk transfer.
9. Contingency plan.
10. Project manager.
11. Quality Assurance team.
12. SQA plan.
13. Audit.
14. Defect tracking.
15. Continuous quality improvement.

Assignments

1. Explain risk evaluation in software projects. Discuss risk identification, probability, impact, risk exposure, qualitative evaluation, quantitative evaluation, and risk prioritisation with suitable examples.

2. Describe the major risk control strategies. Explain risk avoidance, risk reduction, risk transfer, risk acceptance, contingency planning, and risk monitoring.

3. Discuss the roles and responsibilities of the project manager, risk coordinator, development team, QA team, and stakeholders in software risk management.

4. Explain the objectives and major components of an SQA plan. Discuss the importance of standards, procedures, responsibilities, quality metrics, and documentation.

5. Describe SQA execution and monitoring. Explain the roles of reviews, audits, inspections, defect tracking, corrective actions, and quality reporting.

6. Explain continuous quality improvement in software engineering. Discuss how quality measurements, defect analysis, and lessons learned can improve software development processes.

SGOU

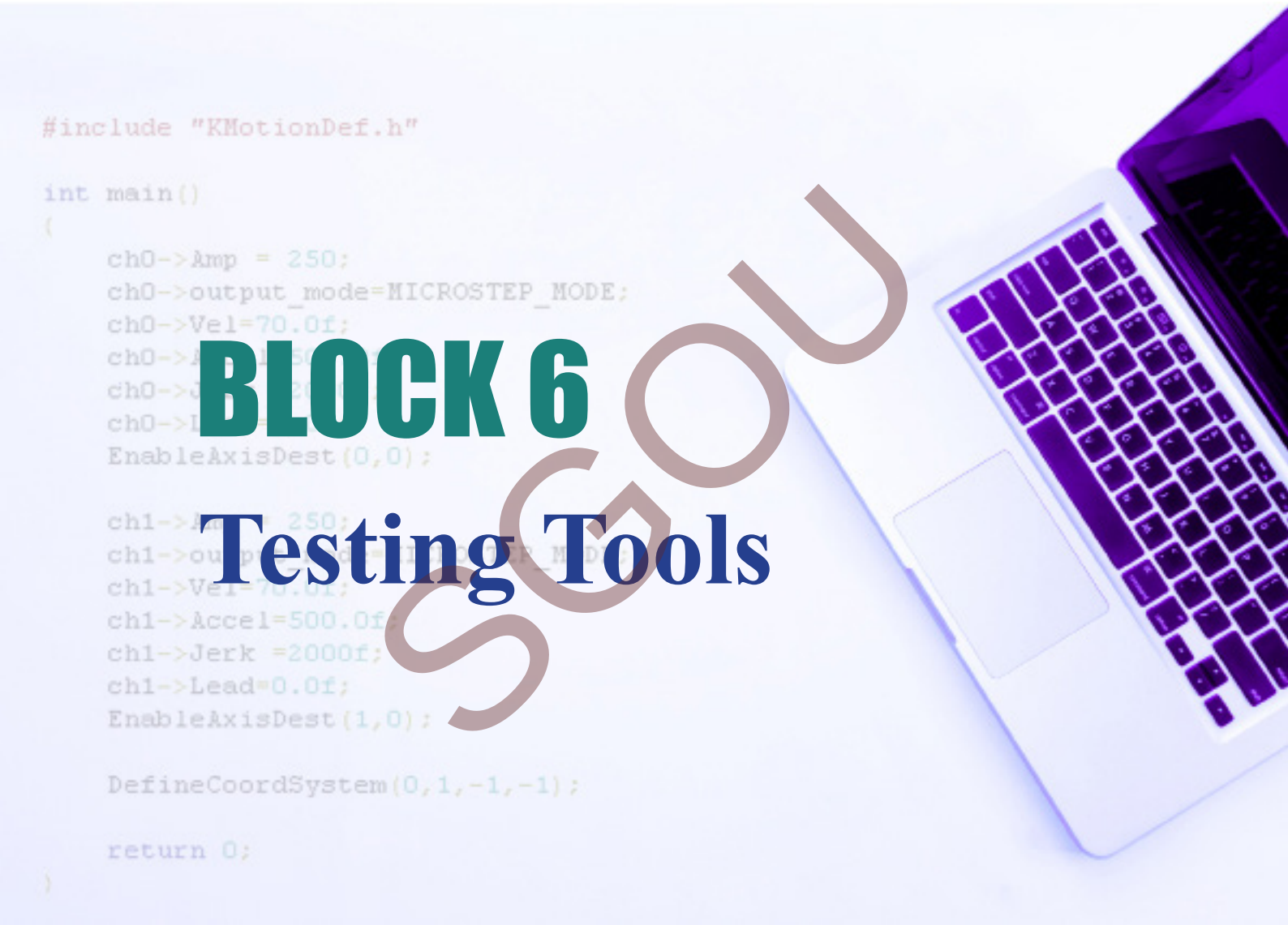
Suggested Reading

1. NIST Risk Management Framework – NIST Risk Management Framework
2. NASA Software Assurance and Software Safety – NASA Software Assurance Resources
3. NASA Software Engineering Handbook – NASA Software Engineering Handbook
4. Resources on software risk identification, assessment, mitigation, and management. https://www.sei.com/capabilities/security-risk-and-compliance/risk-management
5. Guidelines for risk identification, evaluation, treatment, monitoring, and communication. https://www.iso.org/standard/65694.html

Reference

1. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.
2. Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
3. Galin, D. (2018). *Software quality: Concepts and practice*. Wiley-IEEE Press.
4. Goetsch, D. L., & Davis, S. B. (2021). *Quality management for organizational excellence: Introduction to total quality* (9th ed.). Pearson.
5. Wallace, L., Keil, M., & Rai, A. (2004). Understanding software project risk: A cluster analysis. *Information & Management, 42*(1), 115–125.

SGOU



BLOCK 6

Testing Tools

SGOU



UNIT 1 Categories of Testing Tools

Learning Outcomes

After completion of this unit, learner will be able to:

- classify software testing tools
- explain the functions of testing tools
- differentiate testing tool categories
- use testing tools for software testing activities
- select appropriate testing tools for specific testing requirements

Prerequisite

Every day, millions of people rely on software without giving much thought to what happens behind the scenes. A student submits an online examination, a customer transfers money through a banking app, a family books railway tickets, or a shopper places an order during a festive sale. We naturally expect these applications to work correctly, respond quickly, and protect our information. But what if a login suddenly fails, the payment is processed incorrectly, or the website crashes when thousands of users access it at the same time?

Building software is only one part of creating a successful application. Equally important is verifying that every feature works as expected, identifying defects before users encounter them, and ensuring that the system performs reliably under different conditions. As software projects become larger and more complex, carrying out these activities manually becomes difficult. This is why software testers rely on specialized testing tools that help them organize testing activities, record and monitor defects, automate repetitive tests, and evaluate system performance efficiently.

Understanding these tools is an important step toward learning modern software testing. Whether you plan to become a software developer, tester, or project manager, knowing how testing tools support the software development process will help you appreciate how reliable and high quality software is produced.

Let us now begin our study of Testing Tools and their Categories, and explore how different tools contribute to delivering dependable software applications.

SGOU

Keywords

Test Management Tools, Defect Tracking Tools, Functional Testing Tools,Performance Testing Tools

Discussion

6.1.1 Introduction

Have you ever noticed that popular applications such as online shopping websites, banking apps, and social media platforms are updated frequently? Before each update is released, the software must be tested carefully to ensure that everything works correctly. As the number of features increases, managing testing activities becomes challenging. Testers need to organize test cases, record defects, verify software functionality, and evaluate system performance. Performing all these tasks manually can be difficult, especially in large software projects.

This is where testing tools play an important role. Testing tools help improve the efficiency and accuracy of software testing activities. Different tools are designed for different purposes. Some tools help manage testing activities, some track defects, while others assist in functional and performance testing. Understanding these categories of testing tools is essential for selecting the right tool for a specific testing requirement.

In this unit, you will explore the major categories of testing tools, their functions, features, and applications in software testing.

6.1.2 What are Testing Tools?

Before exploring different categories of testing tools, let us first understand what a testing tool is.

Whenever software is developed, it must be tested to ensure that it meets user requirements and performs as expected. Testing involves many activities such as planning test cases, executing tests, recording results, reporting defects, and monitoring software performance. Managing all these activities manually can be time-consuming and may lead to errors.

A **testing tool** is a software application that assists in performing one or more testing activities. These tools help testers work more efficiently, improve accuracy, and reduce the effort required for testing.

SGOU

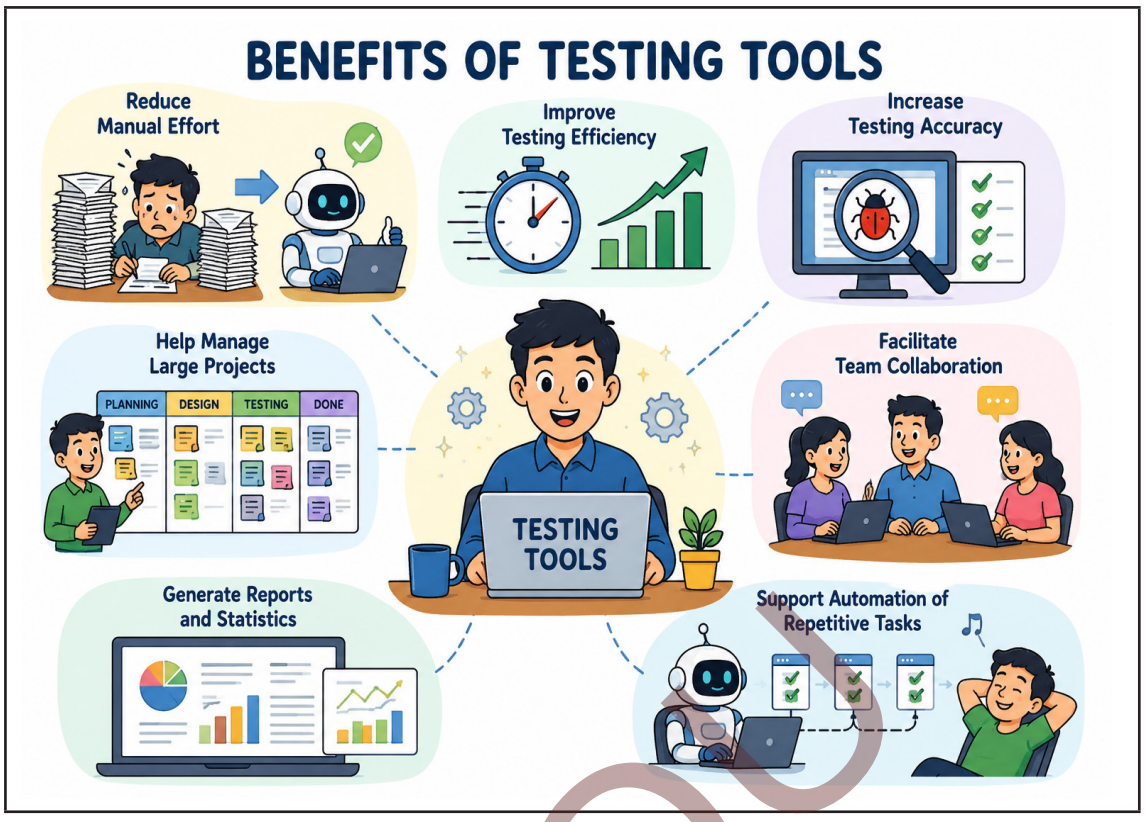


Fig.6.1.1. Benefits of Testing tools

Consider a software project that contains hundreds of test cases and multiple modules. Testers need to know:

- Which test cases have been executed?
- Which test cases have passed or failed?
- How many defects have been reported?
- Which defects have been fixed?
- How does the software perform under heavy usage?

Answering these questions manually can be difficult. Testing tools provide an organized way to manage such information.

Consider the following activities:

1. Recording a software defect
2. Managing hundreds of test cases
3. Measuring website response time

Can a single tool perform all these activities?

SGOU

In practice, different tools are designed for different testing purposes. Therefore, testing tools are classified into various categories based on their functions.

6.1.3 Categories of Testing Tools

Testing tools can be grouped into different categories according to the type of testing activity they support.

The major categories of testing tools are:

1. Test Management Tools
2. Defect Tracking Tools
3. Functional Testing Tools
4. Performance Testing Tools

Each category serves a specific purpose in the software testing process.

Suppose a tester discovers a defect in an online banking application.

What should happen next? The defect should be recorded, the development team should be informed and the defect should be monitored until it is fixed.

A tool designed for such activities belongs to the category of Defect Tracking Tools. Similarly, if the goal is to manage test cases, a Test Management Tool would be more suitable.

Let us now examine each category in detail.

6.1.3.1 Test management Tools

Testing a software application involves much more than simply executing test cases. Test activities must be planned, organized, monitored, and documented.

As software projects become larger, managing these activities manually becomes increasingly difficult. Test management tools provide a systematic approach for handling testing activities.

Test Management Tools are software applications used to plan, organize, manage, and monitor software testing activities. They act as a central repository for all testing-related information. These tools help teams create test plans, design test cases, assign testing tasks, monitor test execution, track testing progress, and generate reports efficiently throughout the software testing process.

A simple example: Consider an educational institution developing an online examination system.

SGOU

The testing team prepares test cases for:

- Student login
- Exam registration
- Question display
- Answer submission
- Result generation

If these test cases are stored in separate documents, managing them can become difficult. A test management tool stores all test-related information in one place, making it easier to organize and track testing activities.

Key Features of Test Management Tools

1. **Test Planning:** Test plans define the scope, objectives, resources, and schedule of testing activities. A test management tool helps create and maintain these plans.

2. **Test Case Management:** The tool allows testers to create, edit, organize, and maintain test cases.

Example:

Test Case ID	Description
TC001	Verify login functionality
TC002	Verify password reset
TC003	Verify logout functionality

3. **Test Execution Tracking:** The status of each test case can be recorded as:Passed/Failed/Blocked/Not Executed. This helps monitor testing progress.

4. **Requirement Traceability:** Requirements can be linked to corresponding test cases. This ensures that every requirement is tested.

5. **Reporting and Dashboard:** Most test management tools provide graphical reports and dashboards showing test execution status, test coverage, defect statistics and project progress

Popular Test Management Tools

Some widely used test management tools include:

1. TestRail
2. Zephyr

SGOU

3. qTest

4. TestLink

6.1.3.2 Defect Tracking Tools

While testing a software application, defects are almost inevitable. A defect may appear as a login failure, an incorrect calculation, a missing button, or even a system crash. Finding a defect is only the beginning of the process. The defect must be documented, assigned to the appropriate developer, monitored during correction, and verified after it has been fixed. When many testers and developers work together on a project, managing defects manually can become confusing. This creates the need for defect tracking tools.

Defect tracking tools are software applications used to record, manage, and monitor software defects throughout their life cycle. These tools provide a centralized location where defects can be reported, assigned, tracked, and reviewed.

Consider a team developing an online banking application. During testing, a tester discovers that the account balance is not updated correctly after a money transfer. Simply informing the development team verbally may not be sufficient. The issue needs to be documented clearly so that it can be investigated and resolved. A defect tracking tool helps achieve this objective.

Whenever a defect is reported, detailed information is entered into the tool. This information typically includes a unique defect identification number, a description of the problem, the module in which the defect occurred, the severity of the defect, and its current status.

Suppose a software project contains hundreds of defects. Some defects may have already been fixed, some may still be under investigation, while others may be waiting for verification. Without a proper tracking mechanism, important defects can easily be overlooked.

Defect tracking tools help teams answer questions such as:

- How many defects are currently open?
- Which defects have been fixed?
- Which defects are critical?
- Who is responsible for resolving a particular defect?

By providing answers to these questions, defect tracking tools improve communication and coordination among project members.

SGOU

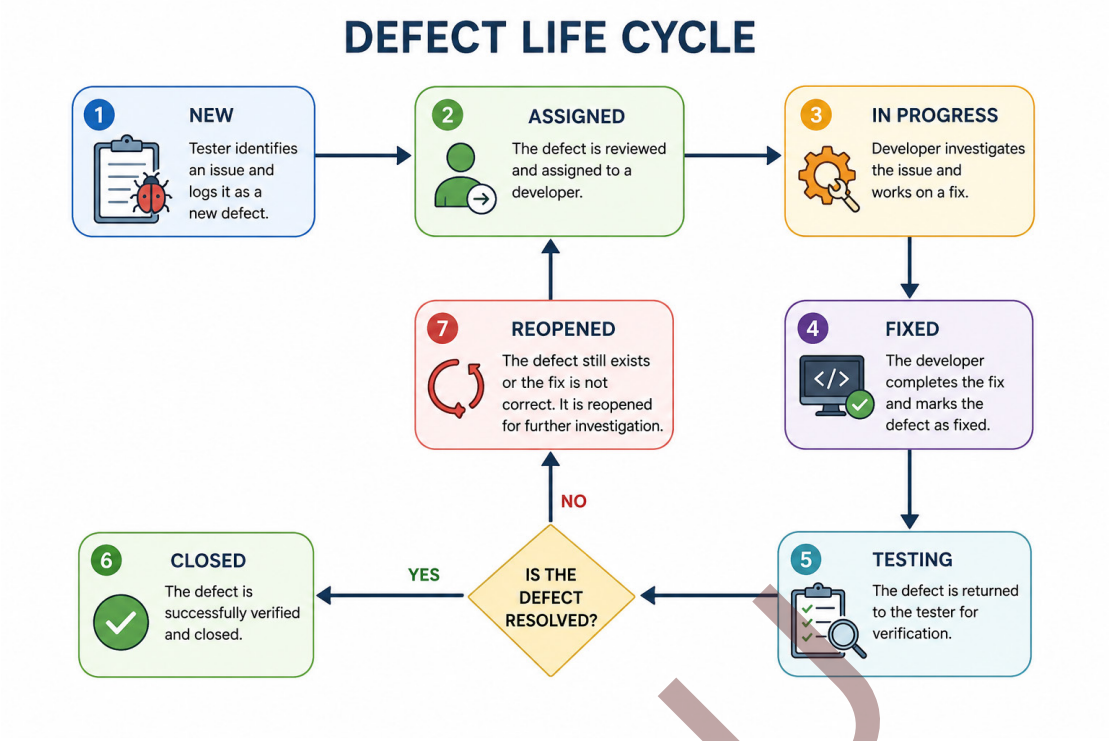


Fig.6.1.2 Defect Life cycle

A defect usually passes through several stages before it is completely resolved. This sequence of stages is known as the defect life cycle.

When a tester discovers a problem, the defect is first recorded as a new defect. After review, it is assigned to a developer for correction. The developer investigates the issue and attempts to fix it. Once the fix is completed, the defect is returned for testing. The tester verifies whether the problem has been resolved successfully. If the defect no longer exists, it is closed. Otherwise, it may be reopened for further investigation.

This systematic process ensures that defects are not forgotten or ignored.

Information Stored in a Defect Report

A defect report contains information that helps developers understand and reproduce the problem. A typical defect report includes the defect identifier, title, description, severity level, priority level, status, and details about the person responsible for resolving the issue.

SGOU

For example, consider the following defect report.

Field	Details
Defect ID	BUG-101
Title	Login failure after password reset
Description	The user is unable to log in after successfully resetting the password.
Module	User Authentication
Severity	High
Priority	High
Status	Open
Assigned To	Development Team
Reported By	Tester
Date Reported	15 July 2025

A well-prepared defect report provides all the information required to understand and reproduce the problem. It enables the development team to identify the root cause, resolve the issue efficiently, and allows testers to verify the fix during retesting. Maintaining complete and accurate defect reports also improves communication among team members and serves as a valuable reference for future software maintenance.

Popular Defect Tracking Tools

Several defect tracking tools are widely used in software organizations. Jira is one of the most popular tools and is commonly used in agile software development environments. Bugzilla is an open-source defect tracking system known for its reliability. Redmine and MantisBT are also widely used for managing software defects.

Although these tools differ in appearance and features, their primary goal remains the same: helping teams manage defects effectively.

6.1.3.3 Functional Testing Tools

Now let us consider another aspect of software testing. Assume that a login page has been developed for a website. How can testers verify that the login feature works correctly with valid and invalid inputs? Furthermore, how can the same tests be repeated every time the software is updated?

Performing these checks manually again and again can be tedious. Functional testing tools help automate and simplify this process.

SGOU

What are Functional Testing Tools?

Functional testing tools are software applications that verify whether a software system performs its intended functions according to the specified requirements. They focus on **what** the software does rather than **how** it is implemented.

In other words, these tools check whether the software produces the expected output for a given input. They do not examine the internal program code. Instead, they interact with the software just as an end user would.

For example, consider an online shopping website. A customer logs in, searches for a product, adds it to the shopping cart, and completes the payment. Each of these activities represents a function of the application. Functional testing tools help verify that every function operates correctly under different conditions.

As software applications evolve, new features are added and existing features are modified. Every modification carries the risk of introducing new defects. Therefore, the same set of tests often needs to be executed repeatedly.

Suppose a banking application has a login module. After adding a new security feature, testers must verify that the login process still works correctly. Executing the same login tests manually after every software update can consume considerable time and effort.

Functional testing tools automate these repetitive tasks. Once a test script has been created, it can be executed multiple times with minimal human intervention. This improves testing efficiency and ensures consistency in the testing process.

How Do Functional Testing Tools Work?

Functional testing tools simulate the actions performed by a user. They enter data, click buttons, navigate through web pages, and compare the actual results with the expected results.

The testing process generally follows these steps.

1. The tester identifies the functionality to be tested.
2. Test cases or automation scripts are prepared.
3. The tool executes the test cases automatically.
4. The actual output is compared with the expected output.
5. A report is generated showing whether each test case has passed or failed.

Consider the login page of a university portal. The tool enters different combinations of usernames and passwords, attempts to log in, and verifies whether the system responds correctly in each case.

SGOU

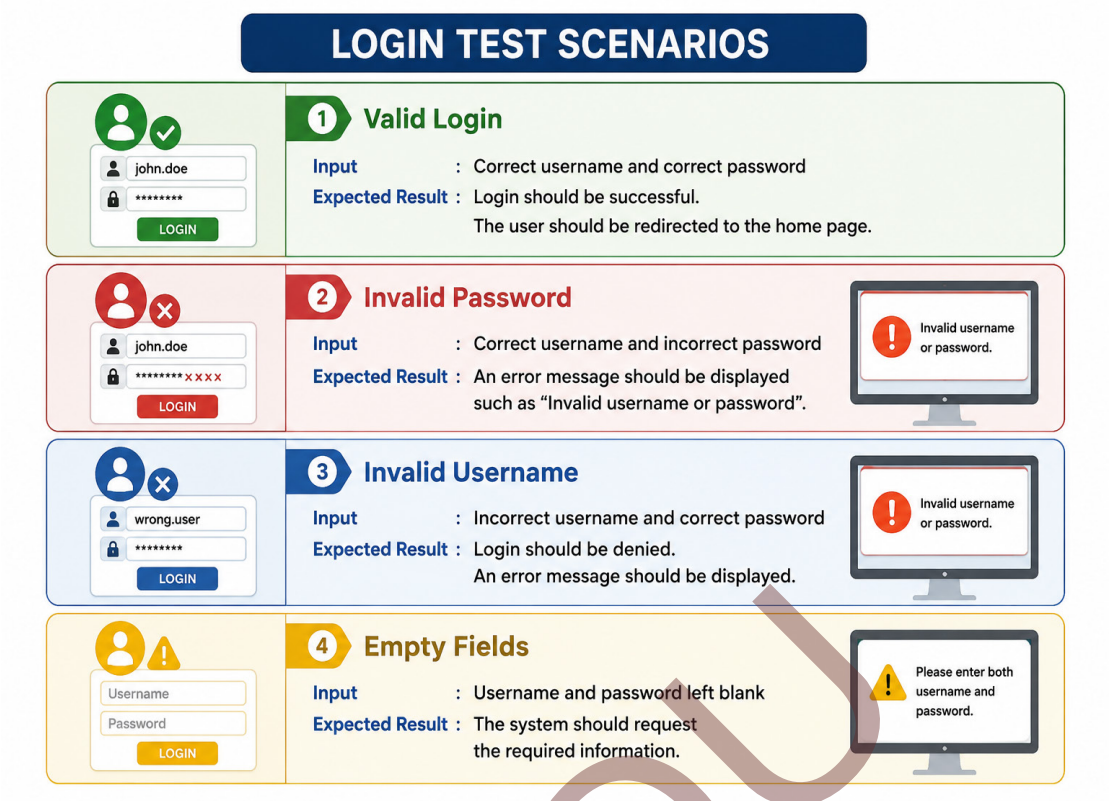


Fig.6.1.3 Functional testing of Login

A functional testing tool executes these test cases automatically and records the results.

Popular Functional Testing Tools

Several functional testing tools are widely used in the software industry. Some of the commonly used tools are listed below.

Tool	Primary Use
Selenium	Automates web application testing across different browsers.
Cypress	Performs fast and reliable testing of modern web applications.
Playwright	Automates testing across multiple browsers and platforms.
UFT (Unified Functional Testing)	Supports functional and regression testing for desktop and web applications.

Each tool has its own strengths. The choice of tool depends on factors such as the type of application, project requirements, programming language support, and the team's expertise.

SGOU

Functional Testing vs. Manual Testing

At this point, you may wonder whether functional testing tools completely replace manual testing. The answer is **No**.

Manual testing is still important for exploratory testing, usability testing, and situations that require human observation and judgment. Functional testing tools are mainly used to automate repetitive and well-defined test cases.

The following table highlights the differences between manual testing and automated functional testing.

Manual Testing	Automated Functional Testing
Test cases are executed by a human tester.	Test cases are executed automatically using software tools.
Suitable for small projects and exploratory testing.	Suitable for repetitive and large-scale testing.
Execution is comparatively slower.	Execution is faster once the scripts are created.
More prone to human error.	Produces consistent and repeatable results.
Requires continuous human effort.	Requires minimal human intervention during execution.

6.1.3.4 Performance testing tools

So far, we have discussed tools that help manage testing activities, track defects, and verify whether software functions correctly. However, another important question still remains.

What happens when thousands of users access the software at the same time?

Think about an online ticket booking system. When ticket booking opens for a popular concert or a major sporting event, thousands of users may try to book tickets simultaneously. Similarly, an online shopping website may experience a sudden surge in visitors during a festive sale. Even if all the functions of the application work correctly, the system may become slow, unresponsive, or even crash when many users access it at once.

How can developers determine whether the software can handle such situations before it is released?

This is where performance testing tools become essential.

Performance testing tools are software applications used to evaluate the speed, stability, responsiveness, and scalability of a software application under different workloads.

SGOU

Unlike functional testing tools, which verify whether a feature works correctly, performance testing tools determine how well the application performs when multiple users access it simultaneously.

For example, consider a college admission portal. During the last date of admission, thousands of students may try to submit their applications at the same time. If the portal responds quickly without crashing, it indicates good performance. If pages take a long time to load or the server becomes unavailable, the application requires performance improvements.

Key Performance Metrics

Performance testing tools measure several important parameters to evaluate the efficiency of an application.

Response Time refers to the time taken by the application to respond to a user's request. A shorter response time generally indicates better performance.

Throughput represents the number of requests processed by the application within a given period. Higher throughput indicates that the system can handle more user requests efficiently.

Concurrency refers to the number of users accessing the application simultaneously.

Resource Utilization indicates how efficiently system resources such as CPU, memory, and network bandwidth are used during execution.

Types of Performance Testing

Performance testing is not limited to measuring speed. Depending on the objective, different types of performance testing are carried out.

Load Testing

Load testing evaluates how the application performs under the expected number of users.

For example, if a university expects 2,000 students to access its examination portal simultaneously, load testing verifies whether the portal performs efficiently under that expected workload.

Stress Testing

Stress testing evaluates the behavior of the application beyond its expected operating capacity.

Suppose the examination portal is designed for 2,000 users but suddenly receives 5,000 users. Stress testing determines whether the system fails gracefully or crashes unexpectedly.

Spike Testing

Sometimes the number of users increases suddenly within a very short time.

SGOU

Consider an online railway reservation system immediately after ticket booking opens. Thousands of users may access the system within a few seconds. Spike testing evaluates how the application responds to such sudden increases in traffic.

Endurance Testing

Some applications run continuously for several hours or even days. Endurance testing verifies whether the application maintains stable performance over a long period.

For example, a banking server operates throughout the day without interruption. Endurance testing helps determine whether memory leaks or performance degradation occur after prolonged execution.

How Do Performance Testing Tools Work?

Performance testing tools simulate the activities of multiple users by generating virtual user requests.

Instead of asking hundreds or thousands of real users to access the application simultaneously, the tool creates virtual users that interact with the application automatically.

A typical performance testing process involves the following steps:

1. Identify the application or module to be tested.
2. Define the expected number of users.
3. Configure the testing scenario.
4. Execute the performance test.
5. Monitor system resources and response time.
6. Analyze the generated reports.

The reports help developers identify performance bottlenecks and optimize the application before deployment.

Popular Performance Testing Tools

Several tools are available for evaluating software performance. Some of the widely used tools are listed below.

Tool	Primary Use
Apache JMeter	Load and performance testing of web applications and services.
LoadRunner	Enterprise-level performance and load testing.
Gatling	Performance testing with detailed analytical reports.
NeoLoad	Automated load testing for web and mobile applications.

Each tool offers different features, and the selection depends on the project's requirements, application type, and available resources.

SGOU

Recap

What Are Testing Tools?

- A testing tool is a software application used for testing.
- Testing tools reduce manual effort.
- Testing tools improve testing accuracy.
- They organize testing information efficiently.
- Different testing activities require different tools.

Categories of Testing Tools

- Testing tools are classified based on their functions.
- The four major categories are:
  - Test Management Tools
  - Defect Tracking Tools
  - Functional Testing Tools
  - Performance Testing Tools

Test Management Tools

- Used to plan and organize testing activities.
- Store all testing information in one place.
- Help create and manage test plans.
- Organize and maintain test cases.
- Track test execution status.
- Provide requirement traceability.
- Generate reports and dashboards.

Popular Tools: TestRail, Zephyr, qTest, TestLink

Defect Tracking Tools

- Used to record and manage software defects.
- Assign defects to developers.
- Track defects throughout their life cycle.
- Improve communication among team members.
- Store complete defect information.

Defect Life Cycle: New → Assigned → Fixed → Retested → Closed (or Reopened)

Popular Tools: Jira, Bugzilla, Redmine, MantisBT

SGOU

Functional Testing Tools

- Verify software functions against requirements.
- Focus on expected outputs.
- Automate repetitive test cases.
- Compare actual and expected results.
- Generate test execution reports.
- Improve testing consistency.
- Do not replace manual testing completely.

Testing Steps:

- Identify functionality.
- Prepare test cases.
- Execute tests.
- Compare results.
- Generate reports.

Popular Tools: Selenium, Cypress, Playwright, UFT

Performance Testing Tools

- Measure software speed and responsiveness.
- Evaluate stability and scalability.
- Simulate multiple users.
- Identify performance bottlenecks.
- Help optimize applications before deployment.

Key Performance Metrics:

- Response Time
- Throughput
- Concurrency
- Resource Utilization

Types of Performance Testing:

- Load Testing
- Stress Testing
- Spike Testing
- Endurance Testing

Testing Steps:

- Identify the application.
- Define the workload.
- Configure the test.
- Execute the test.
- Monitor performance.
- Analyze reports.

Popular Tools: Apache JMeter, LoadRunner, Gatling, NeoLoad

SGOU

Objective Type Questions

1. What is the software application used to assist testing activities called?
2. Which category of testing tools is used to organize testing activities?
3. Which category of testing tools is used to record software defects?
4. Which category of testing tools verifies software functionality?
5. Which category of testing tools evaluates system performance?
6. What is the document that defines the scope and objectives of testing?
7. What is the collection of steps used to verify a software feature called?
8. Which feature links software requirements with test cases?
9. What is the unique identifier assigned to a software defect?
10. What is the first stage of the defect life cycle?
11. What is the final stage of the defect life cycle?
12. Which attribute indicates the impact of a defect on the software?
13. Which attribute indicates the urgency of fixing a defect?
14. Which popular tool is widely used for defect tracking in Agile projects?
15. Which open-source tool is commonly used for defect tracking?
16. Which testing approach verifies software according to user requirements?
17. Which popular tool is widely used for web application test automation?
18. Which testing method requires human intervention?
19. What is the time taken by an application to respond to a request called?
20. What is the number of requests processed by a system in a given time called?
21. What is the number of users accessing an application simultaneously called?
22. What is the efficient use of CPU and memory during execution called?
23. Which type of performance testing evaluates the expected user workload?
24. Which type of performance testing evaluates the system beyond its expected capacity?
25. Which type of performance testing checks the effect of sudden traffic increases?
26. Which type of performance testing evaluates long-duration execution?
27. Which open-source tool is widely used for load testing web applications?
28. Which enterprise-level tool is commonly used for performance testing?
29. Which performance testing tool is known for detailed analytical reports?
30. Which performance testing tool supports automated load testing?

SGOU

Answers to Objective Type Questions

1. Testing Tool
2. Test Management Tool
3. Defect Tracking Tool
4. Functional Testing Tool
5. Performance Testing Tool
6. Test Plan
7. Test Case
8. Traceability
9. Defect ID
10. New
11. Closed
12. Severity
13. Priority
14. Jira
15. Bugzilla
16. Functional Testing
17. Selenium
18. Manual Testing
19. Response Time
20. Throughput
21. Concurrency
22. Resource Utilization
23. Load Testing
24. Stress Testing
25. Spike Testing
26. Endurance Testing
27. Apache JMeter
28. LoadRunner
29. Gatling
30. NeoLoad

SGOU

Assignments

1. Explain the major categories of testing tools. Describe the purpose, key features, and applications of Test Management Tools, Defect Tracking Tools, Functional Testing Tools, and Performance Testing Tools with suitable examples.

2. Discuss the role of Defect Tracking Tools in software testing. Explain the defect life cycle and describe the information typically included in a defect report.

3. Explain the importance of Performance Testing Tools in software development. Describe the different types of performance testing and the key performance metrics used to evaluate software performance.

4. A software company is developing an online university admission portal. The testing team needs to:

- Manage hundreds of test cases.
- Record and monitor software defects.
- Verify that the admission process works correctly.
- Ensure that the portal can support 10,000 students accessing it simultaneously.

Identify the appropriate category of testing tool for each requirement and justify your choice.

5. A shopping website works correctly during normal usage but becomes slow and unresponsive during a festive sale when thousands of customers access it simultaneously.

SGOU

Suggested Reading

1. Graham, D., Black, R., Van Veenendaal, E., & Evans, I. (2020). *Foundations of software testing: ISTQB certification* (5th ed.). Cengage Learning.
2. Crispin, L., & Gregory, J. (2014). *More agile testing: Learning journeys for the whole team*. Addison-Wesley Professional.
3. Dustin, E., Garrett, T., & Gauf, B. (2009). *Implementing automated software testing: How to save time and lower costs while raising quality*. Addison-Wesley Professional.
4. Fewster, M., & Graham, D. (1999). *Software test automation: Effective use of test execution tools*. Addison-Wesley.
5. Black, R. (2017). *Managing the testing process* (3rd ed.). John Wiley & Sons.

Reference

1. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.

2. Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.

3. Patton, R. (2005). *Software testing* (2nd ed.). Sams Publishing.

4. Jorgensen, P. C. (2014). *Software testing: A craftsman's approach* (4th ed.). CRC Press.

5. Kaner, C., Falk, J., & Nguyen, H. Q. (1999). *Testing computer software* (2nd ed.). John Wiley & Sons.

SGOU



Automated Testing Tools

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the purpose of software testing tools.
- identify the features of Selenium and other testing tools.
- differentiate automated, cross-browser, load testing, and defect tracking tools.
- describe the role of testing tools in software quality.
- summarize the applications of software testing tools.

Prerequisite

Every day, millions of people use websites and mobile applications to shop online, book tickets, transfer money, attend online classes, and access government services. These applications are expected to work smoothly, respond quickly, and remain available even when thousands of users access them simultaneously. A delay of just a few seconds or an unexpected system failure can frustrate users and lead to financial losses or damage to an organization's reputation.

Developing software with the required features is only one part of building a successful application. It is equally important to verify that the application performs reliably under different workloads and that any problems discovered during testing are recorded and resolved systematically. Modern software development therefore relies on specialized testing tools that help automate testing activities, evaluate application performance, and manage software defects efficiently.

In this unit, you will explore some of the most widely used testing tools, including Selenium for automated testing, LoadRunner for load testing, and Jira for defect tracking. Understanding these tools will help you appreciate how software organizations improve software quality, reduce testing effort, and deliver reliable applications to users. Let us now begin our study of modern software testing tools.

SGOU

Keywords

Automated testing, Selenium, Cross-Browser testing, Load testing, LoadRunner, Defect tracking

Discussion

6.2.1 Introduction

You may have used an online shopping website to purchase your favourite products. Imagine shopping during a festive sale. In one situation, the website looks perfect in one browser, but the 'Add to Cart' button does not work in another browser. In another situation, thousands of customers visit the website at the same time, causing the pages to load slowly or even crash before you can complete your purchase. In yet another situation, you successfully add products to your cart and make the payment, but the order is not placed because of a software defect. Such problems can frustrate customers, reduce their trust in the website, and cause financial losses for the company. These situations highlight the importance of software testing in ensuring that websites function correctly, perform efficiently, and provide a smooth user experience.

Developing software is only the first step in creating a successful application. Before software reaches users, it must be tested thoroughly to ensure that it functions correctly, performs efficiently, and provides a consistent experience across different environments. As software applications continue to grow in size and complexity, manual testing alone is no longer sufficient. Repeating the same tests after every software update can consume significant time and effort.

This challenge has led to the development of specialized software testing tools. These tools assist testers in automating repetitive tasks, verifying software compatibility across different browsers, evaluating system performance under heavy workloads, and managing software defects systematically. Instead of replacing human testers, these tools make testing faster, more accurate, and more reliable.

Today, almost every software organization uses testing tools as part of its software development process. Whether the application is a banking system, an online learning platform, a hospital management system, or a social media application, testing tools play an important role in delivering high-quality software.

This unit introduces four widely used categories of testing tools:

- Automated Testing Tools
- Cross-Browser Testing Tools
- Load Testing Tools
- Defect Tracking Tools

SGOU

Each category serves a different purpose, and together they help improve software quality throughout the testing process.

Think About It

A software company develops an online examination portal. Before releasing the portal, the testing team wants to answer the following questions.

- Can repetitive tests be executed automatically?
- Does the portal work correctly in Chrome, Firefox, and Edge?
- Can the portal support thousands of students during examinations?
- How can software defects be recorded and monitored?
- Would a single testing tool be able to perform all these tasks?

The answer is **No**. Different testing activities require different testing tools. Understanding the purpose of each tool helps testers choose the most appropriate one for a given situation.

6.2.1.1 Why Do We Need Different Testing Tools?

Software testing involves several activities. Each activity has its own objectives and challenges.

Consider an online shopping application. One tester wants to verify that the Login button works correctly. Another tester wants to check whether the website displays properly in different web browsers. A performance engineer wants to know whether the website can handle 20,000 customers during a festive sale. A project manager wants to monitor defects reported by different testers.

Although all of them are involved in software testing, their objectives are different. Therefore, they require different software testing tools.

The following table illustrates this idea.

Table 6.2.1 Different Testing Activities and Suitable Tools

Testing Activity	Suitable Tool Category
Automating repetitive test cases	Automated Testing Tool
Checking browser compatibility	Cross-Browser Testing Tool
Evaluating system performance	Load Testing Tool
Recording and monitoring defects	Defect Tracking Tool

Selecting the appropriate testing tool improves testing efficiency and reduces manual effort.

SGOU

6.2.2 Automated Testing Tools

Software applications are updated regularly. New features are added, existing features are modified, and software defects are corrected. After every update, testers need to verify that the application continues to work correctly.

Now consider an online banking application.

A tester performs the following tasks every day.

- Opens the login page.
- Enters the username and password.
- Clicks the Login button.
- Checks whether the home page appears.
- Logs out of the application.

Suppose the application is updated every week. Should the tester repeat the same steps manually after every update?

The answer is Yes, because every modification may introduce new defects. However, performing the same tests repeatedly is time-consuming and increases the possibility of human errors.

This is where automated testing becomes valuable.

What is Automated Testing?

Automated testing is the process of using software tools to execute test cases automatically with little or no human intervention.

Instead of performing every step manually, testers create automation scripts that instruct the testing tool to perform the required actions. Once created, these scripts can be executed repeatedly whenever the software changes.

For example, an automation script can automatically Open a web browser, enter login credentials, click the Login button, verify whether the login is successful and record the test result.

The same script can be executed hundreds of times without rewriting it. Automated testing is particularly useful for repetitive testing activities, regression testing, and applications that undergo frequent updates. Fig. 6.2.1 illustrates the process of automated testing.

SGOU

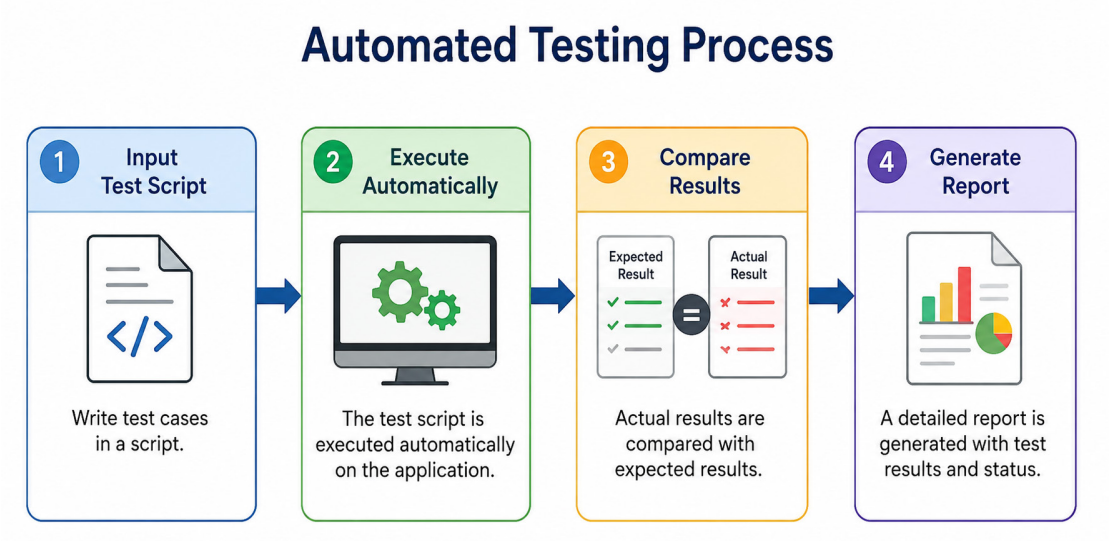


Fig. 6.2.1 Automated testing process

Automated testing offers several advantages over manual testing, especially in large software projects. It helps reduce repetitive work because test cases can be executed multiple times without repeating manual steps. It increases testing speed by executing large numbers of test cases within a short period. It improves consistency because the same test is executed in exactly the same way every time. It reduces human errors that may occur during repetitive manual testing. It supports continuous software development by allowing tests to be executed whenever new software versions are released.

For these reasons, automated testing has become an essential part of modern software development.

6.2.2.1 Selenium

Among the various automated testing tools available today, **Selenium** is one of the most popular and widely used tools for testing web applications. It is an open-source tool that allows testers to automate interactions with a web browser, reducing the need to perform repetitive testing manually.

Suppose a company develops an online shopping website. Before releasing a new version, the testing team needs to verify that customers can log in, search for products, add items to the shopping cart, and complete payments successfully.

If these tests are performed manually after every update, the testing process becomes slow and repetitive. Selenium allows these activities to be automated so that the same tests can be executed repeatedly with consistent results.

Because of its flexibility and browser support, Selenium has become one of the most preferred automation tools in the software industry.

Selenium is an open-source automated testing tool used to test web applications by simulating user actions in a web browser.

SGOU

Instead of asking a tester to perform every action manually, Selenium executes predefined instructions automatically. These instructions are written as automation scripts.

For example, Selenium can perform the following actions automatically.

- Open a web browser.
- Navigate to a website.
- Enter text into input fields.
- Click buttons and links.
- Select options from menus.
- Verify page content.
- Close the browser.

From the user's point of view, it appears as though a real person is interacting with the application.

6.2.2.2 Features of Selenium

Selenium provides several features that make it suitable for automated web testing.

Open Source

Selenium is available free of cost. Organizations can use it without purchasing a software license.

Supports Multiple Browsers

The same automation script can be executed in different browsers such as Chrome, Firefox, Microsoft Edge, and Safari.

Supports Multiple Programming Languages

Automation scripts can be written using several programming languages, including Java, Python, C#, and JavaScript.

Cross-Platform Support

Selenium works on different operating systems such as Windows, Linux, and macOS.

Reusable Test Scripts

Automation scripts created once can be reused multiple times.

Large User Community

Since Selenium is widely used across the world, extensive documentation, tutorials, and community support are available.

SGOU

6.2.2.3 Components of Selenium

Selenium is not a single software application. It consists of several components, each designed for a specific purpose.

The major components are:

1. Selenium IDE
2. Selenium WebDriver
3. Selenium Grid

Each component performs a different function.

1. Selenium IDE

Selenium IDE (Integrated Development Environment) is a tool in the Selenium Test Suite used to create and execute automated test cases. It is commonly known as a record-and-playback tool because it records the actions performed in a web browser and plays them back whenever required.

What does Selenium IDE record?

It records almost every action performed in the browser, such as: Opening a web page, Entering text into input fields, Clicking buttons and links, Selecting options from lists, Navigating between web pages etc. Once these actions are recorded, the test case can be executed repeatedly without performing the steps manually.

How is a test case created?

The process is simple:

1. Open Selenium IDE.
2. Start recording.
3. Perform the required actions on the website.
4. Stop the recording.
5. Save the test case.
6. Run the test case whenever needed.

During execution, Selenium IDE repeats the recorded actions in the same sequence. If the website behaves as expected, the test is successful. Otherwise, Selenium IDE reports the failure, making it easier to identify problems.

Selenium IDE is available as an extension for Google Chrome and Mozilla Firefox. It also allows users to edit recorded test cases, debug them, and group multiple test cases into test suites for organized testing.

Another useful feature is the ability to export recorded test cases to programming languages such as Java, Python, C#, and Ruby, as well as Selenese, Selenium's own scripting language. These exported scripts can be used for advanced automation with Selenium WebDriver.

SGOU

Because Selenium IDE requires little or no programming knowledge for basic test creation, it is an excellent tool for beginners who are learning automated web application testing.

2. Selenium WebDriver

Although Selenium IDE is useful for simple automation, complex software applications require greater flexibility.

This is provided by Selenium WebDriver.

WebDriver allows testers to write automation programs using programming languages such as Java or Python.

Instead of simply recording actions, WebDriver directly communicates with the web browser and performs testing operations.

For example, WebDriver can:

- Read information displayed on a webpage.
- Compare actual and expected values.
- Perform calculations.
- Execute conditional statements.
- Repeat operations using loops.
- Generate detailed reports.

Because of its flexibility, WebDriver is the most widely used Selenium component.

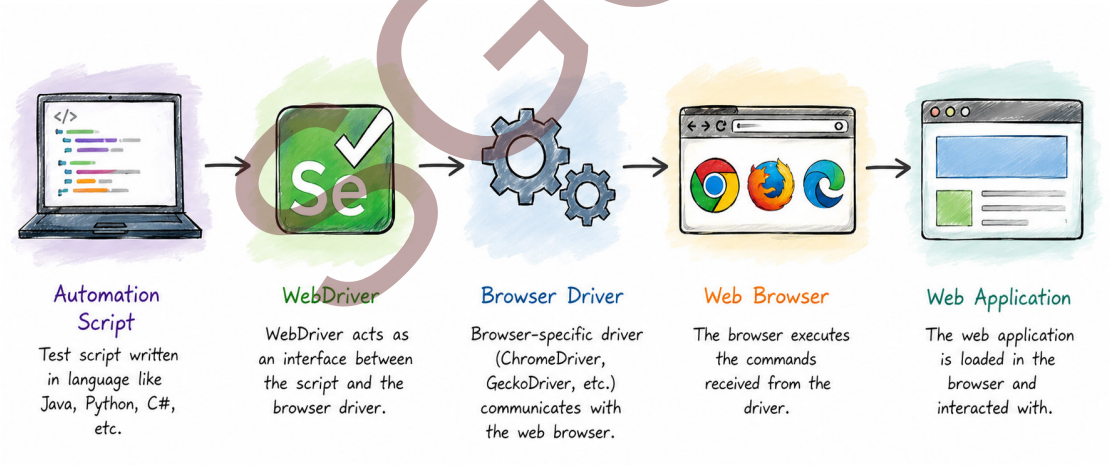


Fig. 6.2.2 Selenium WebDriver Architecture

3. Selenium Grid

Selenium Grid allows multiple tests to run simultaneously on different computers, browsers, and operating systems. This is known as parallel testing. Parallel execution significantly reduces testing time.

For example, a company has 500 automated test cases. If each test requires one minute, executing them sequentially would take approximately 500 minutes. Using

Selenium Grid, several tests can run at the same time, reducing the total execution time considerably.

Selenium is widely used for automating the testing of web applications in software development. It helps testers perform regression testing, verify user interfaces, and test various types of web-based systems such as online shopping websites, banking applications, educational portals, and healthcare applications. Selenium also integrates well with Continuous Integration (CI) environments, enabling automated testing as part of the software development process.

Selenium is a free and open-source automation tool that supports multiple browsers, operating systems, and programming languages. Automation scripts can be reused across multiple test cycles, making regression testing faster and more efficient. By reducing manual effort and ensuring consistent execution of test cases, Selenium improves the overall quality and reliability of software testing.

Selenium is designed primarily for testing web applications and does not directly support desktop application testing. Developing and maintaining automation scripts requires programming knowledge, which may be challenging for beginners. In addition, whenever the user interface of a web application changes, the existing test scripts often need to be updated, increasing maintenance effort.

Note: Selenium is an automation tool, not a complete testing solution. It automates the execution of test cases, but test planning, designing test cases, analyzing results, and exploratory testing still require human involvement.

6.2.3 Cross - browser testing tools

Websites often appear differently when opened in different web browsers. A button may appear correctly in one browser but shift to a different position in another. A menu may work perfectly in one browser but fail to open in another. Although the website is the same, users may experience it differently depending on the browser they use.

Different web browsers use different rendering engines and may interpret HTML, CSS, and JavaScript differently. As a result, a webpage that works correctly in one browser may not behave the same way in another. Therefore, software testers must verify that a web application performs consistently across multiple browsers before it is released.

This process is known as cross-browser testing.

Cross-browser testing is the process of verifying that a web application functions correctly and appears consistently across different web browsers, browser versions, operating systems, and devices. Its primary objective is to ensure that every user experiences the application in the same way, regardless of the browser being used.

For example, a university develops an online admission portal. Students may access the portal using Chrome, Firefox, Edge, Safari, or mobile browsers. Cross-browser testing ensures that all students can complete the admission process without browser-related problems.

SGOU

6.2.3.1 Key aspects tested in Cross-Browser testing

- **Visual Appearance:** Ensures that the layout, fonts, colours, images, and page design appear correctly and consistently across different web browsers.
- **Functionality:** Verifies that buttons, links, forms, menus, and other interactive features work properly in every supported browser.
- **Performance:** Checks whether web pages load quickly and respond efficiently across different browsers and browser versions.

6.2.3.2 Popular Frameworks and Tools

Cross-browser testing can be performed using automation frameworks and cloud-based testing platforms. Automation frameworks help testers create and execute test scripts, while cloud platforms provide access to different browsers and devices without requiring organizations to maintain their own testing infrastructure.

1. Testing Frameworks

Playwright

Playwright is a modern automation framework developed by Microsoft for testing web applications. It supports major browser engines such as Chromium, Firefox, and WebKit (Safari), allowing the same test script to run across multiple browsers. Playwright is known for its speed, reliability, and built-in support for handling multiple browser tabs, user interactions, and network operations.

Cypress

Cypress is a JavaScript-based end-to-end testing framework designed specifically for modern web applications. It allows developers and testers to write, execute, and debug test cases directly within the browser. Cypress provides real-time execution, automatic waiting for page elements, and detailed error messages, making it easier to identify and fix issues during testing.

Selenium

Selenium is one of the most widely used open-source frameworks for automating web application testing. It supports multiple programming languages, including Java, Python, C#, and JavaScript, and works with major web browsers such as Chrome, Firefox, Edge, and Safari. Selenium is highly suitable for regression testing and is commonly integrated with Continuous Integration (CI) tools to automate software testing.

2. Cloud Testing Platforms

BrowserStack

BrowserStack is a cloud-based testing platform that enables testers to execute web applications on thousands of real browsers, operating systems, and mobile devices. It eliminates the need to set up and maintain physical testing devices. BrowserStack supports both manual and automated testing, helping teams verify application compatibility across different environments.

SGOU

LambdaTest

LambdaTest is a cloud platform that supports both manual and automated cross-browser testing. It allows testers to execute automation scripts on a wide range of browser and operating system combinations. The platform also provides features such as parallel test execution, visual testing, and AI-powered test analytics to improve testing efficiency.

Sauce Labs

Sauce Labs is an enterprise-level continuous testing platform that supports testing on real devices, virtual machines, and cloud-based browser environments. It enables parallel execution of multiple test cases, reducing testing time for large projects. The platform also provides detailed logs, screenshots, video recordings, and debugging tools to help identify and resolve issues quickly.

Cloud-based testing platforms have become increasingly popular because they reduce hardware costs, simplify testing across multiple environments, and enable faster software delivery. Together, automation frameworks and cloud testing platforms help organizations perform efficient, reliable, and scalable cross-browser testing.

6.2.4 Load Testing Tools

Introduction to Load Testing

Many web applications experience periods during which a large number of users access them at the same time. An online shopping website may receive thousands of visitors during a festive sale. A university examination portal becomes busy when students log in to download hall tickets or check their results. Similarly, an online banking application may receive a large number of transactions during salary credit days. In all these situations, users expect the application to respond quickly and complete their tasks without interruption.

Now consider an online shopping website that functions perfectly when only a few customers are browsing. Customers can search for products, add items to the shopping cart, and complete payments without any difficulty. However, as the number of users increases, the website begins to respond slowly. Pages take longer to load, payment transactions are delayed, and eventually the website stops responding. Although the software functions correctly, its performance is no longer satisfactory because it cannot handle the increased workload.

Why does this happen?

Every software application is designed to handle a certain amount of workload. When the number of users increases, the server must process more requests simultaneously. If the application's resources are insufficient or its design is inefficient, performance gradually decreases. Users may experience slow response times, incomplete transactions, or server errors. Such problems can reduce customer satisfaction and damage the reputation of an organization.

SGOU

Testing only the functionality of an application is therefore not enough. Developers must also verify whether the application can continue to perform efficiently under its expected workload before it is released to users. This requirement leads to an important type of performance testing known as load testing.

Load testing is the process of evaluating the performance of a software application under the expected number of users or transactions. It helps determine whether the application can maintain acceptable performance when many users access it simultaneously. During load testing, specialized software tools generate virtual users that behave like real users by sending requests to the application. The application's performance is then observed and measured under this workload.

Unlike functional testing, which checks whether a feature works correctly, load testing focuses on how efficiently the application performs while serving multiple users. The objective is not to find functional defects but to identify performance issues that may appear when the system is under normal operating conditions.

Consider a university admission portal expected to receive applications from nearly 5,000 students on the last day of admission. Before the portal is made available, developers simulate 5,000 virtual users accessing the website simultaneously. If the portal continues to respond quickly and processes all requests successfully, it is considered capable of handling the expected workload. If the response time becomes too high or the server stops responding, developers identify the cause of the problem and optimize the application before deployment.

By identifying performance issues in advance, load testing helps organizations deliver reliable software that provides a smooth experience even during busy periods.

6.2.4.1 How Load Testing Works

One obvious question arises at this point.

How can testers evaluate the performance of an application with thousands of users when it is impossible to ask thousands of people to access the application simultaneously?

The answer lies in the use of virtual users.

Load testing tools create virtual users, sometimes called simulated users, that behave like real users. Each virtual user performs actions such as opening web pages, entering login credentials, searching for information, submitting forms, or completing online transactions. Since these actions are performed automatically, hundreds or even thousands of virtual users can interact with the application at the same time.

Suppose an online shopping website is expected to receive 10,000 customers during a festive sale. Instead of inviting 10,000 people to test the website, a load testing tool generates 10,000 virtual users. These virtual users send requests to the application just as real customers would. Some users may search for products, others may add items to the shopping cart, while some proceed to payment. The testing tool records how the application responds throughout the test.

SGOU

While the test is running, the tool continuously monitors the application's performance. It measures how quickly each request is processed, how many requests are completed within a given period, and how efficiently the server uses its available resources such as processor time, memory, and network bandwidth.

If the application begins to slow down as the number of users increases, the testing tool captures this information. Developers can then examine the generated reports to identify the exact point at which the application's performance starts to decline. This enables them to locate performance bottlenecks and improve the application's efficiency before it is released.

The load testing process generally follows a sequence of activities.

First, the application or module to be tested is identified. Next, the expected workload is determined based on the anticipated number of users or transactions. The testing scenario is then configured, specifying how many virtual users should be created and what activities they should perform. After the configuration is complete, the load testing tool executes the test by generating virtual users. During execution, the tool continuously monitors the application's performance and records various performance measurements. Finally, detailed reports are generated, allowing developers to analyze the results and make improvements wherever necessary.

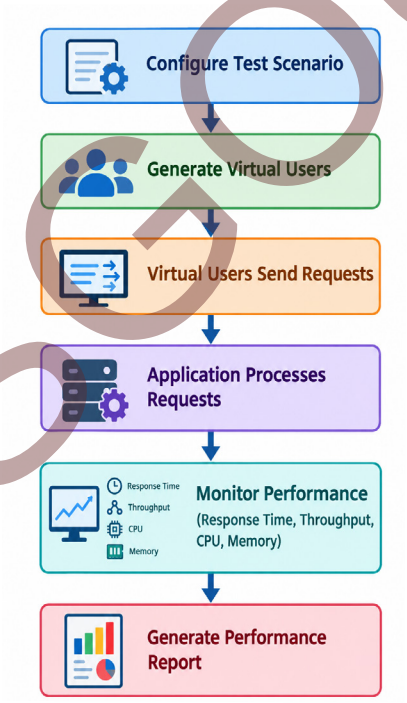


Fig.6.2.3 Working of Load testing

This systematic approach enables organizations to evaluate application performance without disrupting actual users.

6.2.4.2 Key performance Metrics

During load testing, simply observing whether an application works is not enough. Testers need measurable values that indicate how well the application performs under

the given workload. These values are known as performance metrics. They help developers understand the efficiency of the application and identify areas that require improvement.

Suppose two online shopping websites are tested with 5,000 users. Both websites remain operational throughout the test. Does this mean that both applications perform equally well?

Not necessarily.

One website may respond within two seconds, while the other may take ten seconds to display the same page. Similarly, one website may process more customer requests within a minute than the other. Therefore, testers use different performance metrics to evaluate software performance accurately.

Some of the most important performance metrics are response time, throughput, concurrency, and resource utilization.

Response Time

Whenever a user performs an action, such as clicking the Login button or submitting an online form, the application takes some time to process the request and return the result. This duration is called the response time.

In simple terms, response time is the time taken by the application to respond to a user's request.

For example, when a student clicks the **Submit** button after completing an online examination, the system should quickly display a confirmation message. If the confirmation appears within two or three seconds, users generally consider the application to be responsive. However, if they have to wait for twenty or thirty seconds, they may assume that the system has stopped working.

Lower response times usually indicate better application performance.

Throughput

While response time measures the speed of individual requests, another important question remains.

How many requests can the application process within a given period?

The answer is measured by throughput.

Throughput represents the number of requests successfully processed by the application during a specific period.

For instance, consider two banking servers. Server A processes 800 customer requests every minute, whereas Server B processes 2,000 requests during the same period. Since Server B processes more requests, it has a higher throughput.

Higher throughput generally indicates that the application can handle more user activity efficiently.

SGOU

Concurrency

Applications designed for public use often serve many users simultaneously. The number of users accessing the application at the same time is known as concurrency.

For example, an online railway reservation website may have thousands of users booking tickets simultaneously. Similarly, an online admission portal may receive applications from thousands of students on the final day of registration.

Load testing evaluates whether the application can continue to operate efficiently as the number of concurrent users increases.

Resource Utilization

Every application consumes system resources while it is running. These resources include the processor (CPU), memory (RAM), storage devices, and network bandwidth.

Resource utilization indicates how efficiently these resources are used during application execution.

Suppose the processor remains almost fully occupied even when only a few users access the application. This may indicate inefficient program design or insufficient hardware resources. Similarly, excessive memory consumption may eventually reduce application performance.

By monitoring resource utilization during load testing, developers can identify which system resources require optimization.

6.2.4.3 Popular Load Testing Tools

Several software tools are available for evaluating the performance of web applications. Although these tools differ in features and licensing models, they all have a common objective: to simulate multiple users and measure application performance under load.

Among the numerous load testing tools available today, Apache JMeter and LoadRunner are two of the most widely used.

1. Apache JMeter

Apache JMeter is an open-source load testing tool developed by the Apache Software Foundation. It is widely used to evaluate the performance of web applications, web services, databases, and various network-based applications.

Being open source, JMeter can be downloaded and used without purchasing a software license. This has made it popular among students, educational institutions, software developers, and organizations of all sizes.

JMeter works by creating virtual users that interact with an application in much the same way as real users. These virtual users send requests to the application, allowing testers to observe how the application behaves under different workloads.

SGOU

For example, suppose an online examination portal is expected to serve approximately 4,000 students simultaneously. Using JMeter, testers can create 4,000 virtual users who log into the portal, download question papers, submit answers, and log out. During the test, JMeter records response times, throughput, and server resource usage.

The generated reports help developers determine whether the application performs efficiently or whether performance improvements are required.

Features of Apache JMeter

Apache JMeter offers several useful features that make it one of the most popular load testing tools.

It supports testing of web applications, web services, databases, FTP servers, and other network-based applications. It can simulate a large number of virtual users and generate detailed reports in both graphical and tabular formats. JMeter also supports plugins that extend its capabilities, enabling organizations to customize the tool according to their testing requirements.

Another advantage of JMeter is its user-friendly graphical interface, which allows testers to configure testing scenarios without extensive programming knowledge.

Example

A software company develops an online library management system. Before making the application available to students, the testing team uses Apache JMeter to simulate 3,500 users searching books and issuing online requests simultaneously.

After analysing the generated report, the developers observe that the response time increases sharply after 3,000 users. Further investigation reveals that the database requires optimization. After improving the database queries, the system performs efficiently even under the expected workload.

This example illustrates how JMeter helps developers identify performance bottlenecks before software deployment.

2. LoadRunner

Another widely used load testing tool is **LoadRunner**. Unlike Apache JMeter, which is open source, LoadRunner is a commercial performance testing tool designed for enterprise-level applications. It is widely used by large organizations to evaluate the performance, scalability, and reliability of software systems before deployment.

Consider a multinational bank launching a new online banking application. Thousands of customers may log in simultaneously to check account balances, transfer funds, and pay bills. Before making the application available to customers, the organization must ensure that the system can handle this expected workload without slowing down or failing. LoadRunner helps testers evaluate the application's performance under such conditions.

LoadRunner works by creating a large number of virtual users that imitate the actions of real users. These virtual users perform activities such as logging in, searching for infor-

SGOU

mation, submitting forms, and completing transactions. As the virtual users interact with the application, LoadRunner continuously monitors its performance and records important performance metrics such as response time, throughput, and resource utilization.

After the test is completed, LoadRunner generates detailed reports and graphical analyzes that help developers identify performance bottlenecks. These reports make it easier to determine whether the application requires optimization before deployment.

One of the major strengths of LoadRunner is its ability to simulate thousands of virtual users, making it suitable for testing large enterprise applications. It also supports a wide range of application technologies, including web applications, mobile applications, cloud-based systems, databases, and enterprise software. This versatility has made LoadRunner a preferred choice for organizations that develop large-scale software systems.

Recap

Automated Testing Tools

- Automated testing reduces repetitive manual work.
- Selenium automates web application testing.
- Selenium supports multiple web browsers.
- Selenium works with different programming languages.
- Selenium IDE records and replays test cases.
- Selenium WebDriver automates browser interactions.
- Selenium Grid enables parallel test execution.
- Automated testing improves testing consistency.
- Automation saves time in regression testing.
- Selenium is an open-source testing tool.

Cross-Browser Testing Tools

- Different browsers may display web pages differently.
- Cross-browser testing ensures browser compatibility.
- Applications should behave consistently across browsers.
- Browser compatibility improves user experience.
- BrowserStack supports testing on real browsers and devices.

SGOU

- LambdaTest provides cloud-based browser testing.
- Sauce Labs supports automated cross-browser testing.
- Cloud platforms reduce the need for physical test devices.
- Cross-browser testing detects browser-specific issues.
- Consistent browser performance improves software quality.

Load Testing Tools

- Load testing evaluates performance under expected workloads.
- Virtual users simulate real user activities.
- Response time measures application speed.
- Throughput indicates request processing capacity.
- Concurrency represents simultaneous users.
- Resource utilization measures CPU, memory, and network usage.
- Apache JMeter is an open-source load testing tool.
- LoadRunner is an enterprise-level load testing tool.
- Load testing identifies performance bottlenecks.
- Load testing improves application reliability.

Defect Tracking Tools

- Defect tracking manages software defects systematically.
- Every defect follows a defined life cycle.
- Defects are assigned, monitored, and verified.
- Defect reports record complete defect details.
- Severity indicates the impact of a defect.
- Priority determines the order of fixing defects.
- Jira is a popular defect tracking tool.
- Defect tracking improves team communication.
- Proper defect management enhances software quality.
- Accurate defect reports support software maintenance.

SGOU

Objective Type Questions

1. Which tool is widely used for automating web application testing?
2. Which Selenium component executes automation scripts?
3. Which Selenium component supports parallel test execution?
4. Which testing verifies application compatibility across browsers?
5. Which cloud platform provides testing on real browsers and devices?
6. Which cloud-based tool supports Selenium-based cross-browser testing?
7. Which tool is commonly used for enterprise-level cross-browser testing?
8. Which type of testing evaluates software under expected user load?
9. What are simulated users called in load testing?
10. Which metric measures the time taken to respond to a request?
11. Which metric indicates the number of requests processed per unit time?
12. Which metric represents the number of simultaneous users?
13. Which open-source tool is widely used for load testing?
14. Which commercial tool is widely used for enterprise load testing?
15. Which tool is commonly used for defect tracking in agile projects?
16. What is the unique identifier assigned to a software defect called?
17. Which attribute indicates the impact of a defect?
18. Which attribute determines the order in which a defect should be fixed?
19. What is the first stage in the defect life cycle?
20. Which document records the details of a software defect?

SGOU

Answers to Objective Type Questions

1. Selenium
2. WebDriver
3. Grid
4. Cross-browser testing
5. BrowserStack
6. LambdaTest
7. Sauce Labs
8. Load testing
9. Virtual users
10. Response time
11. Throughput
12. Concurrency
13. JMeter
14. LoadRunner
15. Jira
16. Defect ID
17. Severity
18. Priority
19. New
20. Defect report

SGOU

Assignments

1. Explain the role of automated testing tools in software testing. Describe the components and features of Selenium.

2. What is load testing? Explain how load testing is performed and discuss the role of LoadRunner in evaluating software performance.

3. What are defect tracking tools? Explain the defect life cycle and describe how Jira helps in managing software defects.

4. A university plans to launch an online examination portal expected to serve 15,000 students simultaneously. As a software tester, analyze why load testing is essential before deployment and explain how LoadRunner can help identify performance issues.

5. An e-commerce website works correctly in Google Chrome but displays layout issues and non-functional buttons in Mozilla Firefox and Safari. Analyze the possible reasons for these problems and explain how cross-browser testing tools can help ensure a consistent user experience.

Suggested Reading

1. Crispin, L., & Gregory, J. (2014). *More agile testing: Learning journeys for the whole team*. Addison-Wesley Professional.

2. Fewster, M., & Graham, D. (1999). *Software test automation: Effective use of test execution tools*. Addison-Wesley.

3. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.

4. Whittaker, J. A. (2009). *Exploratory software testing: Tips, tricks, tours, and techniques to guide test design*. Addison-Wesley Professional.

Reference

1. Graham, D., Van Veenendaal, E., Black, R., & Evans, I. (2020). *Foundations of software testing: ISTQB certification* (5th ed.). Cengage Learning.

2. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

3. Ammann, P., & Offutt, J. (2017). *Introduction to software testing* (2nd ed.). Cambridge University Press.

4. Mathur, A. P. (2013). *Foundations of software testing* (2nd ed.). Pearson.

SGOU



Mobile and API Testing

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the concepts and importance of mobile and API testing
- identify the tools used for mobile testing, API testing, and mobile application performance testing
- differentiate between mobile device emulators and simulators
- describe the methods used for mobile compatibility testing and API testing
- analyze the role of mobile and API testing in ensuring application quality

Prerequisite

Every day, millions of people use mobile applications to communicate, shop, learn, travel, and manage their finances. Most users simply tap a button and expect the application to respond immediately with accurate information. When checking a bank balance, booking a movie ticket, tracking a food delivery, or viewing the latest weather update, the required information appears within seconds. But have you ever wondered how this information reaches the mobile application?

Behind the scenes, the mobile application continuously exchanges information with remote servers. Whenever a user logs in, searches for a product, uploads a photograph, or makes an online payment, requests are sent to a server, which processes them and returns the appropriate response. This communication is made possible through **Application Programming Interfaces (APIs)**.

Now consider a situation where a food delivery application opens successfully, but no restaurants are displayed. The user interface appears normal, yet the application cannot provide the expected information. Similarly, a banking application may display its login screen correctly but fail to authenticate users because the server does not return the correct response. In such situations, simply testing the user interface is not sufficient. The communication between the application and the server must also be tested carefully.

API testing helps verify that this communication is accurate, reliable, secure, and efficient. It ensures that requests are processed correctly, responses are returned promptly,

SGOU

and the application continues to function smoothly even when many users access the service simultaneously. To perform these tasks efficiently, developers and testers use specialized API testing tools that simplify testing, automate repetitive activities, and identify defects at an early stage of development.

In this section, you will learn the fundamentals of API testing, understand why it is important in mobile application development, and become familiar with some of the widely used API testing tools and their applications. By the end of this section, you will appreciate how API testing contributes to building reliable, efficient, and high-quality mobile applications.

Keywords

Mobile Testing, Performance Testing, Compatibility Testing, Emulator, Simulator, API Testing, Postman, Appium

Discussion

6.3.1 Introduction

Mobile applications have transformed the way people communicate, learn, work, shop, travel, and access services. Whether it is booking a train ticket, transferring money through a banking application, attending an online class, or ordering food, mobile applications have become an indispensable part of everyday life. As the number of mobile users continues to grow, expectations regarding the quality of these applications have also increased. Users expect applications to open quickly, respond instantly to their actions, display information accurately, and operate without interruptions. Even a small delay or an unexpected error may discourage users from continuing to use an application.

Developing a mobile application involves much more than writing program code. An application may appear to function correctly during development but may behave differently when installed on another device or used under different conditions. For example, an application that performs smoothly on a high-end smartphone may become slow on a device with limited memory. Similarly, an application that works well with a strong Wi-Fi connection may experience delays when used on a slower mobile network. Such variations make testing an essential stage of mobile application development.

Testing is a systematic process of evaluating software to ensure that it satisfies specified requirements and performs reliably under different operating conditions. It helps identify defects before the application reaches users, thereby reducing maintenance costs and improving customer satisfaction. Testing also verifies that the application delivers a consistent experience across a wide variety of devices, operating systems, screen sizes, and network environments.

SGOU

Another important aspect of modern mobile applications is their communication with remote servers. Most mobile applications do not work in isolation. A social networking application retrieves user profiles from a server, an online shopping application obtains product details from a database, and a banking application exchanges financial information securely with banking systems. This communication takes place through **Application Programming Interfaces (APIs)**. If the API fails to respond correctly, the mobile application may not function as expected even though its user interface appears normal. Consequently, testing APIs has become as important as testing the mobile application itself.

This unit introduces the concepts of mobile testing and API testing. It explains the purpose of mobile testing, familiarizes learners with commonly used testing tools, discusses performance testing and compatibility testing, distinguishes between emulators and simulators, and describes the importance of API testing and API testing tools. Understanding these concepts provides learners with a strong foundation in modern software testing practices.

6.3.2 Mobile Testing

When software was developed primarily for desktop computers, testing mainly focused on verifying whether the application functioned correctly on a limited number of hardware configurations. The rapid growth of smartphones has changed this situation considerably. Today, mobile applications are expected to operate on thousands of devices manufactured by different companies, each having its own hardware characteristics, operating system versions, display sizes, and processing capabilities. As a result, testing mobile applications has become both more important and more challenging.

Mobile testing is the process of evaluating a mobile application to ensure that it performs its intended functions correctly, provides a satisfactory user experience, and operates reliably across different mobile devices and environments. It examines not only whether the application's features work correctly but also whether the application remains stable, performs efficiently, responds quickly, and adapts to varying operating conditions.

Consider a student using a learning management application. The student expects to log in without difficulty, view course materials clearly, submit assignments successfully, and attend online classes without interruptions. If the application crashes while uploading an assignment or freezes during an examination, the consequences may be serious. Such situations highlight why thorough testing is essential before releasing an application to users.

Mobile testing is therefore not limited to identifying programming errors. It is a comprehensive activity that examines every aspect of an application's behaviour from the perspective of its users. The ultimate objective is to deliver an application that performs reliably regardless of the device, operating system, or network being used.

SGOU

6.3.2.1 Why is Mobile Testing important?

The popularity of mobile applications has created intense competition among software developers. Users have thousands of alternatives available in app stores. If an application performs poorly, users can uninstall it within seconds and switch to another application offering similar services. Consequently, maintaining high software quality has become essential for the success of mobile applications.

Testing contributes to software quality in several ways. It helps detect defects during the development process, allowing developers to correct them before the application reaches users. This reduces the cost of maintenance after release and prevents negative user experiences. Testing also ensures that different features of the application work together correctly, reducing the possibility of unexpected failures during actual use.

Performance is another important consideration. Mobile users expect applications to respond immediately. Delays in loading screens, processing transactions, or displaying information often result in dissatisfaction. Through appropriate testing, developers can identify performance bottlenecks and optimize the application's behaviour.

Security is equally significant. Many mobile applications process sensitive information such as passwords, banking details, personal photographs, and health records. Testing verifies whether appropriate security mechanisms are in place to protect user data against unauthorized access and cyber threats.

Finally, mobile testing helps build user confidence. Applications that function consistently and reliably encourage users to continue using them and recommend them to others. Thus, effective testing contributes directly to customer satisfaction and the overall success of software products.

Think about it …

Have you ever uninstalled a mobile application because it was slow, displayed errors, or stopped responding?

What quality issues influenced your decision?

Reflect on your experience before continuing.

6.3.2.2 Challenges in Mobile Testing

Testing mobile applications presents several practical challenges. As mobile technology continues to evolve, testers must ensure that applications maintain consistent quality despite rapid changes in devices, operating systems, and user expectations.

One of the primary challenges is the **large number of mobile devices** available in the market. Every year, manufacturers introduce new smartphones with improved hardware and software capabilities. Screen sizes range from compact mobile phones to large tablets and foldable devices. Since users may install the same application on any of these devices, testers cannot rely on testing a single model alone.

SGOU

The frequent release of **operating system updates** creates another challenge. Android and iOS regularly introduce new versions containing additional features, security improvements, and interface modifications. While these updates enhance user experience, they may also affect the behaviour of existing applications. Therefore, applications should be evaluated on supported operating system versions to ensure continued compatibility.

Testing also becomes challenging because of **changing network conditions**. Mobile users frequently move between different environments such as homes, educational institutions, shopping centres, and public transport. During these movements, network quality may change continuously. Applications should continue operating smoothly despite fluctuations in signal strength or temporary network interruptions.

Battery consumption is another important concern. Mobile users expect applications to function for long periods without rapidly draining battery power. Applications that continuously access GPS services, cameras, or background processes may consume excessive energy. Testing therefore evaluates whether the application utilizes device resources efficiently.

Storage limitations present an additional challenge. Many users install numerous applications on their smartphones, leaving limited free storage space. A mobile application should install successfully, update correctly, and continue functioning even when storage availability becomes low.

Security has become one of the most significant concerns in mobile computing. Banking applications, healthcare systems, educational platforms, and digital payment services handle highly confidential information. Testers must verify that authentication mechanisms, encryption techniques, and data storage methods adequately protect user information against unauthorized access.

Finally, modern users expect frequent software updates introducing new features and improvements. Every update carries the possibility of introducing new defects into previously functioning components. Consequently, applications undergo repeated testing throughout their development and maintenance lifecycle.

Although these challenges increase the complexity of mobile testing, systematic testing strategies and specialized testing tools help developers deliver reliable and high-quality applications.

6.3.3 Mobile Testing Tools

Testing a simple application manually may not require sophisticated software tools. However, modern mobile applications often contain hundreds of screens, numerous user interactions, and complex communication with servers. Repeating the same testing activities after every software update becomes both time-consuming and error-prone. Consequently, software organizations make extensive use of **mobile testing tools** to improve the efficiency and reliability of the testing process.

A mobile testing tool is a software application that assists testers in planning, executing, monitoring, and documenting testing activities. Some tools automate repetitive testing

SGOU

tasks, while others measure application performance, identify defects, or evaluate compatibility across multiple devices. These tools reduce manual effort and improve the accuracy of testing results.

It is important to understand that no single testing tool performs every testing activity. Different tools are designed for different objectives. Some specialize in functional testing, whereas others focus on performance evaluation or compatibility testing. Therefore, software organizations often use a combination of tools depending on the nature of the application being developed.

Before discussing individual tools, it is useful to understand the common features that most mobile testing tools provide.

A typical mobile testing tool enables testers to automate repetitive test cases, execute tests on multiple devices, record test execution, compare expected and actual results, identify failures, generate detailed reports, and integrate testing activities into the software development process. These capabilities significantly reduce testing time while improving software quality.

In the following sections, some of the widely used mobile testing tools are introduced. Rather than memorizing every feature of these tools, learners should focus on understanding the purpose for which each tool is commonly used and the situations in which it becomes most effective.

6.3.3.1 Common Mobile Testing Tools

As mobile applications became more complex, software developers needed reliable ways to test applications repeatedly and efficiently. Manual testing is useful for observing user experience and identifying visible defects, but repeating the same test cases after every software update can be time-consuming. To solve this problem, mobile testing tools are used to automate testing activities, execute tests on multiple devices, and generate reports about application quality.

Each mobile testing tool has a specific purpose. Some tools are designed for testing Android applications, while others support iOS applications. Certain tools focus on functional testing, whereas others are useful for performance testing or cross-platform testing. Understanding the purpose of these tools helps learners appreciate how software professionals choose appropriate tools for different testing requirements.

Appium

Appium is one of the most widely used open-source tools for mobile application automation testing. It supports testing of Android, iOS, and hybrid mobile applications. Because Appium allows testers to write test scripts in programming languages such as Java, Python, JavaScript, and Ruby, it is considered flexible and suitable for cross-platform testing.

A major advantage of Appium is that it can automate user interactions such as tapping buttons, entering text, scrolling screens, and verifying displayed results. This helps testers execute the same test cases repeatedly without performing each action manually.

SGOU

Major features of Appium

- Supports Android, iOS, and hybrid applications.
- Works with multiple programming languages.
- Allows automation of user interface interactions.
- Supports testing on real devices as well as emulators.
- Is open source and widely used in the software industry.

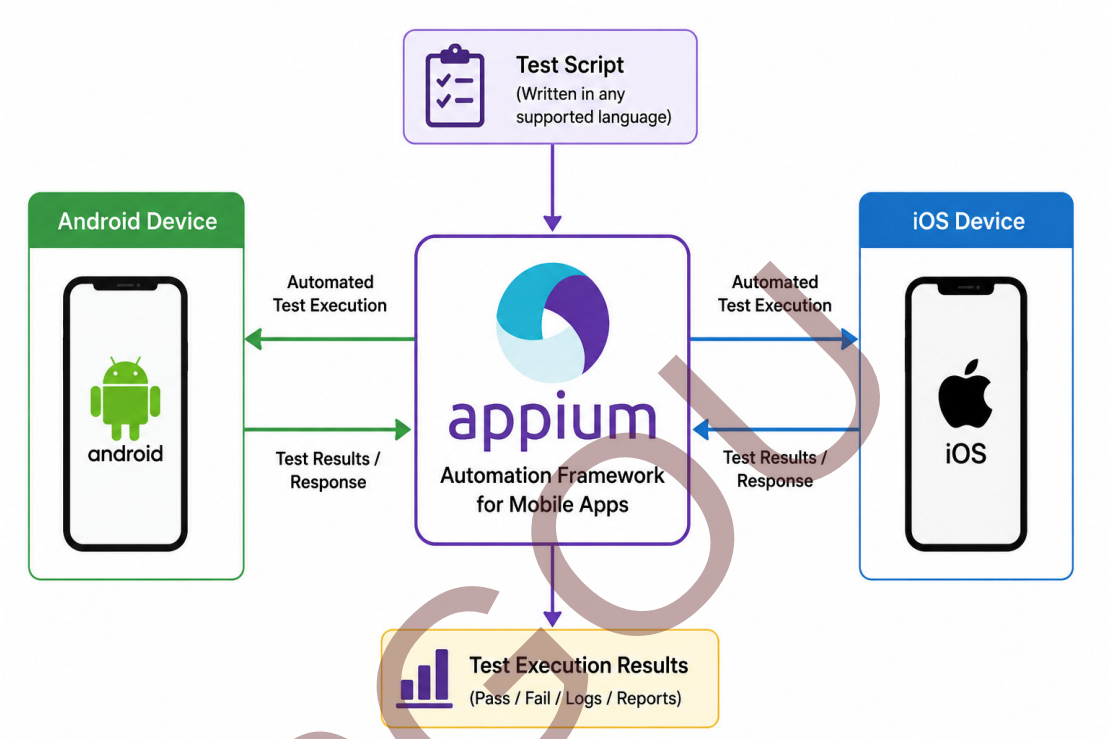


Fig.6.3.1 Appium-Based Mobile Test Automation

Espresso

Espresso is a testing framework developed by Google for Android application testing. It is mainly used for testing the user interface of Android applications. Espresso allows testers to write concise test scripts that verify whether buttons, menus, text fields, and other interface elements behave as expected.

Espresso is particularly useful for developers and testers working with Android Studio because it integrates well with the Android development environment. It is commonly used for testing Android applications at the user interface level.

Major features of Espresso

- Designed specifically for Android applications.
- Provides fast execution of user interface tests.
- Integrates well with Android Studio.
- Allows verification of screen elements and user actions.
- Helps identify interface-related defects.

SGOU

XCUITest

XCUITest is Apple's official framework for testing iOS applications. It is used to automate user interface testing for iPhone and iPad applications. Since it is developed by Apple, XCUITest integrates closely with Xcode, the primary development environment for iOS applications.

XCUITest allows testers to simulate user actions such as tapping buttons, entering text, scrolling through screens, and verifying displayed information. It is commonly used for testing the behaviour and performance of iOS applications.

Major features of XCUITest

- Designed specifically for iOS applications.
- Integrates with Apple's Xcode environment.
- Supports automation of user interface testing.
- Allows verification of screen elements and navigation.
- Is useful for testing iPhone and iPad applications.

Robot Framework

Robot Framework is an open-source automation framework that supports acceptance testing and robotic process automation. Although it is not limited to mobile testing, it can be integrated with tools such as Appium to automate mobile application testing.

One of the strengths of Robot Framework is its keyword-driven testing approach. This makes test cases easier to read and understand, even for learners who are new to automation testing.

Major features of Robot Framework

- Supports keyword-driven testing.
- Can be integrated with Appium for mobile testing.
- Is open source and extensible.
- Helps create readable and maintainable test cases.
- Supports acceptance testing and automation.

Selendroid

Selendroid is another automation testing tool designed for Android applications. It is based on the Selenium framework and supports testing of native, hybrid, and web applications on Android devices. Selendroid can be used with real devices as well as Android emulators.

SGOU

Although newer tools such as Appium are more commonly used today, Selendroid is still useful for understanding the evolution of Android automation testing.

Major features of Selendroid

- Supports Android application testing.
- Works with native, hybrid, and web applications.
- Supports testing on real devices and emulators.
- It is based on Selenium concepts.
- Can automate Android user interface interactions.

6.3.4 Mobile Application Performance Testing

A mobile application may provide useful features, but users are unlikely to continue using it if it is slow, unresponsive, or unstable. Performance is therefore one of the most important quality characteristics of a mobile application. Users expect applications to open quickly, respond immediately to touch actions, process transactions without delay, and function smoothly even when many users access the application simultaneously.

Mobile application performance testing is the process of evaluating how efficiently a mobile application performs under different workloads and operating conditions. It measures factors such as response time, loading speed, CPU usage, memory consumption, battery usage, and network utilization.

Performance testing helps identify bottlenecks that may reduce application efficiency. For example, an application may take too long to load because it is retrieving large amounts of data from a server. Another application may consume excessive memory, causing it to slow down or crash on devices with limited resources.

Objectives of performance testing

1. Measure application response time.
2. Evaluate loading speed of screens and features.
3. Analyze CPU and memory utilization.
4. Check battery consumption.
5. Verify application stability under heavy workload.
6. Identify performance bottlenecks.

Example

Consider a video streaming application. During performance testing, testers verify whether videos start quickly, continue playing smoothly, and maintain acceptable quality when many users access the service at the same time.

SGOU

Common performance metrics

Metric	Meaning
Response time	Time taken by the application to respond to a user action.
Loading time	Time required for the application or screen to load completely.
CPU usage	Amount of processor resources used by the application.
Memory usage	Amount of RAM consumed by the application.
Battery consumption	Amount of battery power used during application operation.
Throughput	Amount of data processed within a given time.

Performance testing is especially important for applications that handle large numbers of users, process online transactions, stream multimedia content, or continuously communicate with servers. Effective performance testing helps developers optimize applications and provide a better user experience.

Remember, a mobile application that functions correctly but responds slowly may still fail to satisfy users. Performance testing ensures that applications are both functional and efficient.

6.3.4.1 Types of Mobile Application Performance Testing

We have learned that performance testing measures how efficiently a mobile application performs under different conditions. However, different situations require different kinds of performance testing. For example, an application may need to support thousands of users during normal operation, while another application may experience sudden traffic during a festival sale or ticket booking event. To evaluate these situations, testers perform different types of performance testing.

The major types of performance testing are described below.

Load Testing

Load testing evaluates the performance of an application under the expected number of users or transactions. It verifies whether the application maintains acceptable response time and stability during normal operating conditions.

Example: An online banking application is tested with 10,000 users simultaneously checking account balances and transferring money.

Stress Testing

Stress testing determines how the application behaves when the workload exceeds its expected capacity. It helps identify the maximum limit the application can handle before performance degrades or the application fails.

SGOU

Example: An online shopping application designed for 20,000 users is tested with 60,000 users during a festive sale.

Spike Testing

Spike testing evaluates the application's ability to handle sudden and significant increases in user traffic. It also verifies whether the application recovers quickly after the traffic returns to normal.

Example: A ticket booking application experiences a sudden surge of users immediately after reservations open.

Endurance (Soak) Testing

Endurance testing checks whether the application continues to perform efficiently when it runs continuously for an extended period under a normal workload. It helps detect memory leaks and gradual performance degradation.

Example: A music streaming application is tested continuously for twelve hours to observe memory usage and battery consumption.

Volume Testing

Volume testing evaluates application performance when processing very large amounts of data. It verifies whether the application can efficiently store, retrieve, and process large databases.

Example: A hospital management application retrieves patient records from a database containing millions of entries.

Scalability Testing

Scalability testing measures how well an application accommodates increasing numbers of users or transactions by adding additional computing resources.

Example: An online learning application expands from supporting 5,000 learners to 50,000 learners using cloud servers.

Each type of performance testing focuses on a different aspect of application performance. Together, they help developers build mobile applications that are fast, reliable, and capable of delivering a satisfactory user experience under various operating conditions.

6.3.4.2 Mobile Device Emulators and Simulators

Developing a mobile application is only the first step in delivering a quality product. Before an application is released to users, it must be tested on different mobile devices, operating systems, and screen sizes. Purchasing and maintaining a large number of smartphones and tablets for testing is expensive and impractical. To overcome this challenge, developers use mobile emulators and mobile simulators. These software tools imitate mobile devices and enable developers to test applications without requiring every physical device.

SGOU

Although the terms emulator and simulator are often used interchangeably, they have different purposes and capabilities. Understanding these differences helps developers choose the appropriate tool for various testing activities.

Mobile Emulator

Imagine that a developer is creating an Android application. After writing a small section of code, the developer wants to verify whether the application opens correctly and whether a newly added feature works as expected. Installing the application on a physical smartphone after every small modification would consume considerable time. Instead, the developer can launch an Android Emulator on the computer, install the application within seconds, and immediately observe its behaviour.

A **mobile emulator** is a software application that recreates both the hardware and software environment of a mobile device. It enables the operating system to run in an environment that closely resembles an actual smartphone or tablet. Since the emulator reproduces the device hardware, applications behave almost as they would on a real mobile device.

For example, an emulator can simulate processor performance, memory usage, storage, GPS location, camera, battery level, and various sensors. This enables developers to verify not only whether the application functions correctly but also how efficiently it utilizes device resources.

Android application developers commonly use the **Android Emulator** provided with Android Studio to test applications throughout the development process. Different virtual devices can be created with different screen sizes, Android versions, and hardware configurations, allowing applications to be evaluated under multiple conditions without requiring several physical smartphones.

Features of Mobile Emulators

Mobile emulators provide several useful features for application development and testing.

- Recreate both hardware and software environments.
- Support multiple versions of mobile operating systems.
- Allow testing on devices with different screen sizes and resolutions.
- Simulate GPS, camera, sensors, battery, and network conditions.
- Support debugging and application monitoring.
- Reduce the need for multiple physical devices during development.

Advantages of Mobile Emulators

Using emulators offers several benefits during software development.

- They reduce the cost of purchasing multiple devices.
- Applications can be tested repeatedly after every modification.

SGOU

- Different operating system versions can be tested easily.
- Hardware-related features can be simulated.
- Defects can be detected early in the development process.

Limitations of Mobile Emulators

Although emulators provide realistic testing environments, they cannot perfectly reproduce every characteristic of an actual mobile device. Factors such as touch sensitivity, battery behaviour, camera quality, processor performance, and real network conditions may differ from those of physical devices.

Therefore, applications that perform well on an emulator should still be tested on real devices before they are released to users.

Simulator

Consider another situation. A developer wants to verify whether the buttons, menus, icons, and screens of a mobile application appear correctly. There is no need to test hardware components such as the camera or GPS at this stage. The primary objective is to ensure that the application interface functions properly.

In such situations, a **mobile simulator** provides a convenient solution.

A mobile simulator recreates only the **software environment** of a mobile device. Unlike an emulator, it does not imitate the device hardware. Instead, it reproduces the appearance and behaviour of the operating system, allowing developers to test application functionality and user interface design quickly.

Since simulators do not recreate hardware components, they generally execute applications faster and require fewer system resources than emulators.

For example, developers creating iPhone applications often use the **iOS Simulator** to verify screen layouts, menu navigation, and application workflow before testing the application on an actual iPhone or iPad.

Features of Mobile Simulators

Mobile simulators offer several useful capabilities.

- Recreate the software environment of mobile devices.
- Execute applications quickly.
- Support functional and user interface testing.
- Require fewer computing resources.
- Easy to install and configure.

Advantages of Mobile Simulators

Simulators are widely used during the early stages of application development because they provide several advantages.

SGOU

- Faster execution than emulators.
- Suitable for verifying screen layouts and navigation.
- Lower hardware requirements.
- Simplify functional testing.
- Reduce application development time.

Limitations of Mobile Simulators

Since simulators do not reproduce hardware components, they cannot accurately evaluate hardware-dependent features. Testing of battery consumption, GPS functionality, camera operation, processor performance, and device sensors requires either an emulator or a physical mobile device.

Consequently, simulators are primarily used to verify application functionality rather than overall device performance.

Commonly Used Mobile Emulators and Simulators

Several tools are available to assist developers in mobile application testing.

Tool	Platform	Purpose
Android Emulator	Android	Tests Android applications in Android Studio
iOS Simulator	iOS	Tests iPhone and iPad applications
Genymotion	Android	Provides fast virtual Android devices for testing
BlueStacks	Android	Runs Android applications on desktop computers

6.3.5 Mobile Compatibility Testing

A mobile application may perform perfectly during development but fail when installed on another device with a different screen resolution or operating system version. Even though both devices supported the same operating system, differences in screen size, hardware configuration, memory capacity, or software version could affect the application's behaviour.

Mobile users own a wide variety of devices. Some use the latest smartphones with advanced processors and large displays, while others use older devices with limited memory and smaller screens. An application that performs well on only a few devices may receive poor reviews if it fails on many others.

To avoid such problems, developers perform mobile compatibility testing.

Mobile compatibility testing is the process of verifying that a mobile application functions correctly across different devices, operating systems, screen sizes, hardware configurations, and network environments. The objective is to ensure that every user

SGOU

experiences the application in a consistent and reliable manner, regardless of the device being used.

For example, consider a mobile banking application. A customer using a high-end smartphone should be able to log in, transfer funds, and view account details without difficulty. At the same time, another customer using an older smartphone should also be able to perform the same operations successfully. Compatibility testing helps ensure that both users receive a similar experience.

6.3.5.1 Factors Considered During Compatibility Testing

Several factors are examined during compatibility testing to ensure that an application performs consistently across different environments.

Device Compatibility

Mobile applications should function correctly on devices produced by different manufacturers. Variations in hardware design, processor architecture, memory capacity, and device features should not affect the application's performance.

Example: An application should work properly on Samsung, Google Pixel, Xiaomi, OnePlus, and Motorola smartphones.

Operating System Compatibility

Mobile operating systems are updated frequently. New versions introduce additional features and security improvements, while older versions continue to be used by many customers.

Compatibility testing verifies that the application functions correctly on multiple versions of the operating system.

Example: An Android application may be tested on Android 13, Android 14, and Android 15 to ensure consistent behaviour.

Screen Size and Resolution Compatibility

Mobile devices are available with different screen sizes and display resolutions. An application interface that appears attractive on a large screen may not display correctly on a smaller device.

Compatibility testing verifies that buttons, menus, text, images, and navigation elements are displayed properly on different screen sizes and orientations.

Example: A shopping application's product images and checkout page should appear correctly on both smartphones and tablets.

Hardware Compatibility

Many mobile applications interact directly with hardware components such as cameras, microphones, fingerprint sensors, GPS receivers, Bluetooth, and NFC.

SGOU

Compatibility testing verifies that these hardware features operate correctly on supported devices.

Example: A QR code scanning application should access the camera successfully on different smartphone models.

Network Compatibility

Mobile users access applications through different types of network connections, including Wi-Fi, 4G, and 5G networks. Network speed may vary depending on location and signal strength.

Compatibility testing evaluates whether the application performs satisfactorily under different network conditions.

Example: A video streaming application should continue playing videos smoothly even when the network speed changes from Wi-Fi to mobile data.

Browser Compatibility

Some mobile applications provide web-based interfaces that are accessed through mobile browsers. Compatibility testing ensures that these web applications function correctly across different browsers.

Example: A university admission portal should display correctly in Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari on mobile devices.

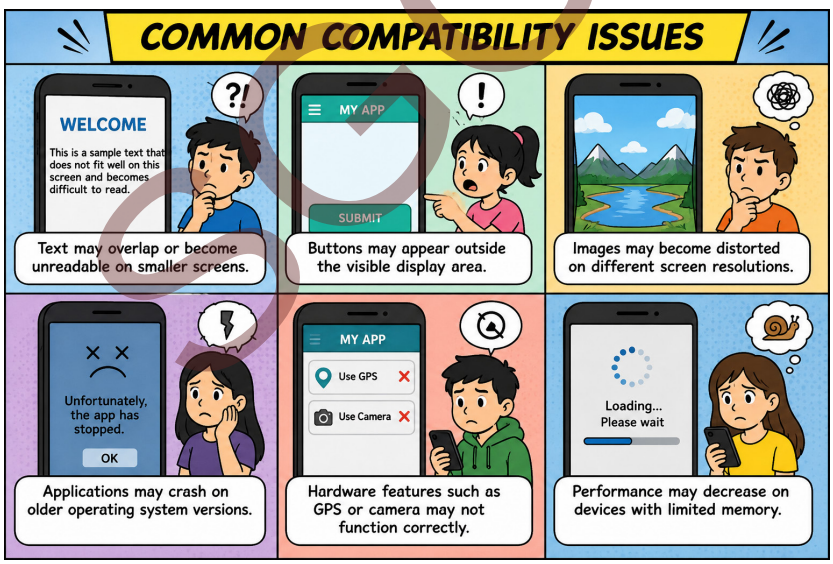


Fig. 6.3.2 Common Compatibility issues

Activity: Identify the type of Compatibility issue

Identify the type of compatibility issue represented in each of the following situations.

1. A mobile application works properly on Android but crashes on iOS.
2. The application interface appears correctly on a tablet but some buttons are hidden on a small smartphone.
3. A QR code scanning application cannot access the camera on certain mobile devices.
4. A video streaming application performs well on Wi-Fi but buffers frequently on a 4G network.
5. A mobile application works correctly on Android 15 but fails on Android 13.
6. A mobile website displays correctly in Google Chrome but not in Safari.
7. An application runs smoothly on devices with higher RAM but becomes slow on devices with lower RAM.
8. A navigation application is unable to detect the user's location on some smartphones.
9. The application layout is displayed properly in portrait mode but becomes distorted in landscape mode.
10. Images and text appear blurred on devices with high-resolution displays.

6.3.6 API Testing Tools

Many mobile applications perform most of their operations by communicating with a server. When a user logs into an application, views products in an online store, books a ticket, or checks a bank balance, the application sends requests to a server and receives the required information in return. This communication takes place through **Application Programming Interfaces (APIs)**.

Now consider the following situation. A mobile shopping application opens successfully, but no products are displayed. The application itself appears to be working, yet the information expected by the user is missing. Is the problem with the mobile application or with the server?

To answer this question, developers test the API separately. By testing the API independently, they can determine whether the server is sending the correct information and whether the application is receiving it correctly.

API testing has therefore become an essential part of mobile application testing.

SGOU

6.3.6.1 What is API Testing?

API (Application Programming Interface) testing is the process of verifying that APIs function correctly by sending requests to them and examining the responses returned by the server. Unlike user interface testing, API testing focuses on the communication between different software components rather than the appearance of the application.

API testing verifies whether the API returns the correct data, responds within an acceptable time, handles invalid requests appropriately, and performs reliably under different conditions.

For example, when a user enters a username and password in a banking application, the application sends these details to the server through an API. The server checks whether the credentials are valid and returns an appropriate response. API testing verifies that this entire communication takes place correctly.

API testing helps developers identify problems at an early stage and improves the reliability of mobile applications.

Some of the important reasons for performing API testing are:

- Verifies that APIs return accurate data.
- Ensures correct communication between the mobile application and the server.
- Detects errors before the application is released.
- Improves application reliability.
- Reduces debugging time.
- Supports faster software development.

6.3.6.2 Common API Testing Tools

Several tools are available to simplify API testing. These tools allow testers to send requests, examine responses, verify data, and automate testing activities.

Postman

Postman is one of the most widely used API testing tools. It provides a simple graphical interface for creating and sending API requests without writing complex programs.

Using Postman, testers can:

- Send GET, POST, PUT, and DELETE requests.
- View API responses.
- Verify response status codes.
- Test request parameters and headers.
- Automate repeated API tests.

SGOU

Example: A tester sends a login request using Postman to verify whether valid user credentials produce a successful login response.

SoapUI

SoapUI is a popular API testing tool that supports both SOAP and REST web services. It is widely used to perform functional, regression, security, and performance testing of APIs. The tool allows testers to automate test execution, create data-driven test cases using different input values, and validate API responses efficiently. Its comprehensive testing capabilities make it well suited for testing enterprise applications that use different types of web services, helping ensure that APIs function correctly, securely, and reliably.

REST Assured

REST Assured is a Java-based library designed specifically for testing RESTful APIs.

Unlike Postman, which provides a graphical interface, REST Assured allows developers to write automated API tests using Java programs.

It is commonly used in continuous integration and automated software testing environments.

Insomnia

Insomnia is another API testing tool that provides an easy-to-use interface for sending API requests and examining server responses.

It supports:

REST APIs

- GraphQL APIs
- Environment management
- Authentication testing
- Response validation

Its simple interface makes it suitable for beginners as well as experienced developers.

Swagger

Swagger is a collection of tools used for designing, documenting, and testing APIs. It allows developers to understand API functions through interactive documentation and execute API requests directly from the documentation page.

Swagger is widely used during API development because it improves communication between developers and testers.

SGOU

Recap

Mobile Testing

- Mobile testing verifies the quality, reliability, and usability of mobile applications.
- It ensures that applications function correctly under different operating conditions.
- Mobile testing improves performance, security, and user experience.
- Testing helps identify defects before application release.
- Mobile testing faces challenges due to device diversity, operating system updates, network variations, and hardware differences.
- Mobile testing tools automate repetitive testing tasks and improve testing efficiency.
- Appium, Espresso, XCUITest, Robot Framework, and Selendroid are widely used mobile testing tools.

Mobile Application Performance Testing

- Performance testing evaluates the speed, efficiency, and stability of mobile applications.
- Response time, loading time, CPU usage, memory usage, battery consumption, and throughput are important performance metrics.
- Load testing evaluates performance under expected workload.
- Stress testing identifies the application's maximum capacity.
- Spike testing evaluates sudden increases in workload.
- Endurance testing measures long-term application stability.
- Volume testing evaluates performance with large amounts of data.
- Scalability testing measures the application's ability to support increasing users.

Mobile Device Emulators and Simulators

- Emulators and simulators reduce the need for multiple physical devices during testing.
- An emulator recreates both the hardware and software environment of a mobile device.
- A simulator recreates only the software environment.
- Emulators support hardware-related and performance testing.
- Simulators are suitable for functional and user interface testing.

SGOU

- Real-device testing is necessary before application release.

Mobile Compatibility Testing

- Compatibility testing ensures consistent application behaviour across different environments.
- It verifies compatibility with devices, operating systems, screen sizes, hardware, networks, and browsers.
- Compatibility testing improves application reliability and user satisfaction.
- Early detection of compatibility issues reduces maintenance costs.

API Testing Tools

- API testing verifies communication between mobile applications and servers.
- It ensures that APIs return correct and timely responses.
- API testing improves application reliability and detects defects early.
- Postman simplifies API request and response testing.
- SoapUI supports SOAP and REST API testing.
- REST Assured automates REST API testing using Java.
- Insomnia supports REST and GraphQL API testing.
- Swagger supports API design, documentation, and testing.

Objective Type Questions

1. Name the testing process that verifies whether a mobile application works correctly on different devices.
2. Identify the testing that measures the speed and efficiency of a mobile application.
3. Name the open-source tool used for automating Android and iOS application testing.
4. Identify Google's testing framework for Android applications.
5. Name Apple's official testing framework for iOS applications.
6. Identify the automation framework that supports keyword-driven testing.
7. Name the Android testing tool based on the Selenium framework.
8. What type of performance testing evaluates an application under the expected workload?

SGOU

9. What type of performance testing determines the application's breaking point?

10. What type of testing evaluates sudden increases in user traffic?

11. What type of testing verifies long-term application stability?

12. What type of testing evaluates application performance with large volumes of data?

13. What type of testing measures an application's ability to support increasing numbers of users?

14. Name the software tool that recreates both the hardware and software environment of a mobile device.

15. Name the software tool that recreates only the software environment of a mobile device.

16. Identify the testing performed to verify application behaviour on different devices and operating systems.

17. Name the compatibility testing performed on different mobile browsers.

18. Identify the compatibility testing that evaluates application performance under different network conditions.

19. Expand the abbreviation API.

20. Name the testing that verifies communication between a mobile application and a server.

21. Identify the API testing tool that provides a graphical interface for sending requests.

22. Name the API testing tool that supports both SOAP and REST web services.

23. Identify the Java-based library used for testing RESTful APIs.

24. Name the API tool used for interactive API documentation.

25. Identify the API testing tool that supports both REST and GraphQL APIs.

SGOU

Answers to Objective Type Questions

1. Compatibility Testing
2. Performance Testing
3. Appium
4. Espresso
5. XCUITest
6. Robot Framework
7. Selendroid
8. Load Testing
9. Stress Testing
10. Spike Testing
11. Endurance Testing
12. Volume Testing
13. Scalability Testing
14. Emulator
15. Simulator
16. Compatibility Testing
17. Browser Compatibility Testing
18. Network Compatibility Testing
19. Application Programming Interface
20. API Testing
21. Postman
22. SoapUI
23. REST Assured
24. Swagger
25. Insomnia

SGOU

Assignments

1. Explain the importance of mobile testing in software development. Discuss the major challenges encountered during mobile application testing.

2. Describe the commonly used mobile testing tools. Compare the features and applications of Appium, Espresso, and XCUITest.

3. Explain the purpose of API testing. Discuss the features and applications of Postman, SoapUI, REST Assured, Insomnia, and Swagger.

4. A mobile banking application works correctly on Android 15 but crashes on Android 13. The login page also displays incorrectly on small-screen devices. Identify the compatibility issues involved and suggest suitable testing approaches to resolve them.

5. A food delivery application becomes slow during lunch hours when thousands of users place orders simultaneously. Explain which types of performance testing should be performed to identify the problem and justify your answer.

Suggested Reading

1. Ammann, P., & Offutt, J. (2017). *Introduction to software testing* (2nd ed.). Cambridge University Press.
2. Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons learned in software testing*. John Wiley & Sons.
3. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.
4. Pettichord, B. (2002). *Lessons learned in software testing: A context-driven approach*. John Wiley & Sons.
5. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Reference

1. Beck, K. (2002). *Test driven development: By example*. Addison-Wesley.
2. Crispin, L., & Gregory, J. (2014). *More agile testing: Learning journeys for the whole team*. Addison-Wesley.
3. Fewster, M., & Graham, D. (1999). *Software test automation: Effective use of test execution tools*. Addison-Wesley.
4. Patton, R. (2005). *Software testing* (2nd ed.). Sams Publishing.
5. Whittaker, J. A. (2009). *Exploratory software testing: Tips, tricks, tours, and techniques to guide test design*. Addison-Wesley.

SGOU



Security and Validation Testing

Learning Outcomes

After completion of this unit, learner will be able to:

- explain the purpose and importance of security testing in web applications.
- identify common web application security vulnerabilities
- describe the role of security testing tools in identifying application vulnerabilities.
- differentiate between static and dynamic security testing approaches.
- explain the purpose of CSS validation and interpret the results generated by a CSS validator tool.

Prerequisite

Every day, millions of people use web applications without giving much thought to what happens behind the scenes. We transfer money through online banking, shop from e-commerce websites, submit assignments through learning portals, book tickets, and connect with others on social media. We simply expect these applications to work correctly and keep our personal information safe.

But what if a hacker could log in to someone else's account without knowing the password? What if a small mistake in a website allowed an attacker to steal customer information or change important records? Surprisingly, many cyberattacks begin with simple weaknesses that developers fail to notice during software development. A single overlooked vulnerability can lead to financial losses, privacy breaches, and loss of user trust.

Building software with attractive features is only one part of developing a successful application. Equally important is ensuring that the application is secure against unauthorized access and malicious attacks. This is achieved through security testing, which helps identify and fix vulnerabilities before they can be exploited. Along with understanding common threats such as SQL Injection and Cross-Site Scripting (XSS), developers also use specialized security testing tools and techniques to strengthen web applications and protect valuable information.

SGOU

In this section, you will learn the fundamentals of security testing, explore common web application vulnerabilities, understand the role of security testing tools, study SQL Injection and XSS testing, compare Static and Dynamic Application Security Testing, and become familiar with CSS validation for developing reliable, standards-compliant web applications. Let us begin by understanding the concept and importance of Security Testing and Security Testing Tools.

Keywords

Security Testing, Web Application Security, Security Vulnerabilities, SQL Injection (SQLi), Cross Site Scripting (XSS), SAST, DAST, CSS Validator

Discussion

6.4.1 Security Testing and Security Testing Tools

Modern software applications store and process a large amount of valuable information. Banking applications manage financial transactions, shopping websites store customer details, educational portals maintain academic records, and social networking platforms handle personal information shared by millions of users. Every day, users trust these applications with sensitive data such as passwords, contact information, payment details, and personal documents.

Now think about this for a moment. What would happen if an attacker gained unauthorized access to an online banking application? Money could be transferred illegally. What if a university website was hacked? Student records could be altered or deleted. If an e-commerce website were attacked, customer payment details might be stolen. Such incidents not only result in financial losses but also damage the reputation of the organization and reduce users' confidence in its services.

These examples highlight the importance of protecting software applications from security threats. Developing an application with many useful features is not enough. It must also be secure against unauthorized access, data theft, and malicious attacks. This is where security testing plays an important role.

Security testing is a software testing technique used to determine whether an application can protect its data, resources, and services from security threats. It evaluates whether the application can prevent unauthorized users from accessing confidential information and verifies that legitimate users can safely perform their intended activities. In other words, security testing helps identify weaknesses before attackers discover and exploit them.

Unlike functional testing, which checks whether the software performs its intended functions correctly, security testing focuses on identifying vulnerabilities that may allow attackers to misuse the application. A software application may function perfectly from the user's perspective but still contain hidden security weaknesses. Security testing aims to discover these weaknesses and help developers correct them before the software is released.

SGOU

The primary objective of security testing is to ensure that the application protects information effectively and maintains the confidentiality, integrity, and availability of data. These three principles are commonly known as the **CIA Triad**, which forms the foundation of information security.

- **Confidentiality** ensures that sensitive information is accessible only to authorized users.
- **Integrity** ensures that data cannot be modified or deleted without proper authorization.
- **Availability** ensures that authorized users can access the system and its services whenever required.

When these three principles are maintained, users can trust that the application is secure and reliable.

Security testing is performed throughout the software development life cycle rather than only after the application has been completed. During development, developers and testers continuously examine the application to identify security weaknesses. Detecting vulnerabilities early reduces development costs and minimizes the risk of security breaches after deployment.

Consider an online shopping website. A customer logs in, browses products, adds items to the cart, and completes payment. During this process, several security-related questions arise.

- Can another user access someone else's account?
- Are passwords stored securely?
- Is payment information protected during transmission?
- Can an attacker manipulate product prices?
- Can unauthorized users access confidential customer information?

Security testing helps answer these questions by evaluating whether the application can resist various forms of attack.

6.4.2 Why is Security testing important?

As technology advances, cyberattacks have become increasingly sophisticated. Attackers continuously search for vulnerabilities in websites and applications to steal information, disrupt services, or gain unauthorized access. Even a small security weakness can lead to serious consequences.

Security testing offers several important benefits

It protects sensitive information such as passwords, financial records, medical information, and personal details from unauthorized access. By identifying vulnerabilities before software deployment, organizations can prevent data breaches and financial losses.

SGOU

Security testing also improves user confidence. When users know that an application protects their information, they are more likely to trust and continue using the service. For businesses, maintaining customer trust is essential for long-term success.

Another important benefit is compliance with legal and industry regulations. Many organizations are required to follow data protection standards and cybersecurity guidelines. Security testing helps ensure that applications satisfy these requirements.

In addition, identifying vulnerabilities during development is generally less expensive than fixing security problems after deployment. Early detection reduces maintenance costs and prevents disruptions to business operations.

6.4.3 Common Security Risks in Web Applications

Although developers carefully design software applications, security weaknesses may still exist. These weaknesses are known as **vulnerabilities**. Attackers attempt to exploit vulnerabilities to gain unauthorized access or disrupt application services.

Some common security risks found in web applications include:

- **Weak authentication:** Applications with weak password policies or insecure login mechanisms make it easier for attackers to gain unauthorized access.
- **Improper authorization:** Sometimes users can access resources or information that should be restricted to administrators or specific users.
- **Poor input validation:** Applications that do not properly validate user input may allow attackers to inject malicious commands or scripts.
- **Sensitive data exposure:** Confidential information such as passwords or credit card numbers may be stored or transmitted without adequate protection.
- **Session management issues:** If user sessions are not properly managed, attackers may hijack active sessions and impersonate legitimate users.

In the following sections of this unit, you will study two of the most common web application attacks **SQL Injection** and **Cross-Site Scripting (XSS)** in detail.

6.4.4 Security Testing Tools

Imagine that a teacher has to evaluate answer sheets for only five students. The task can be completed manually without much difficulty. However, if there are thousands of answer sheets, manual evaluation becomes slow and prone to errors. Similarly, manually examining every part of a large software application for security weaknesses is extremely difficult.

To make security testing faster, more accurate, and systematic, testers use security testing tools.

Security testing tools are software applications designed to identify security vulnerabilities in websites, web applications, mobile applications, and computer

SGOU

systems. These tools automate many testing activities, enabling testers to discover weaknesses that might otherwise remain unnoticed.

Security testing tools perform several important tasks. They scan applications for known vulnerabilities, identify insecure configurations, detect weak authentication mechanisms, examine input validation, and generate detailed reports describing the identified issues. These reports help developers understand the vulnerabilities and implement appropriate corrective measures.

Instead of replacing human testers, security testing tools assist them by automating repetitive tasks and allowing them to focus on analysing complex security issues.

Different tools are designed for different purposes. Some tools scan web applications for vulnerabilities, while others analyze source code, monitor network traffic, or assess the security of computer systems. Selecting an appropriate tool depends on the type of application being tested and the objectives of the security assessment.

Some widely used security testing tools include:

Tool	Purpose
OWASP ZAP	Identifies security vulnerabilities in web applications through automated and manual testing.
Burp Suite	Performs web application security testing by intercepting and analysing HTTP requests and responses.
Acunetix	Automatically scans websites for common security vulnerabilities such as SQL Injection and XSS.
Nessus	Scans computer systems and networks to identify security weaknesses, configuration errors, and missing security updates.

Each tool has its own strengths. Some are suitable for beginners, while others provide advanced capabilities for professional security analysts. Regardless of the tool used, the primary objective remains the same to identify vulnerabilities before attackers can exploit them.

6.4.5 Web Application Security Testing

Today, web applications have become an integral part of our daily lives. People use them for online banking, shopping, ticket booking, education, healthcare, social networking, and many other services. Unlike traditional desktop applications, web applications can be accessed through a web browser from any location with an Internet connection. While this accessibility provides great convenience, it also exposes web applications to various security threats. Attackers constantly search for weaknesses that can be exploited to steal sensitive information, modify data, or disrupt services. Therefore, ensuring the security of web applications has become an essential aspect of software development.

SGOU

Web Application Security Testing is the process of evaluating a web application to identify security vulnerabilities before attackers can exploit them. It verifies whether the application adequately protects sensitive information, authenticates users correctly, controls access to resources, and securely processes user requests. The objective is to identify security weaknesses and recommend corrective measures before the application is deployed or updated.

Unlike functional testing, which verifies whether an application performs its intended functions correctly, web application security testing focuses on protecting the application against unauthorized access, data breaches, and malicious activities. Even if a web application functions perfectly, it cannot be considered reliable if it contains security vulnerabilities.

For example, consider an online banking application. Customers expect to view account balances, transfer money, and pay bills securely. If an attacker manages to bypass the login page or manipulate account details, the consequences can be severe. Similarly, in an online shopping application, customers expect their personal information and payment details to remain confidential. Security testing helps identify such weaknesses before the application becomes available to users.

6.4.5.1 Objectives of Web Application Security Testing

Web application security testing is performed to achieve several important objectives.

The first objective is to protect sensitive information such as usernames, passwords, financial records, and personal details from unauthorized access. Confidential information should only be accessible to legitimate users.

Another objective is to verify user authentication. The application should ensure that only authorized users can log in using valid credentials. Weak authentication mechanisms increase the risk of unauthorized access.

Security testing also evaluates authorization mechanisms. After successful authentication, users should only be able to access resources and perform actions permitted by their roles. For example, a student should not be able to modify examination results, while an administrator may have such privileges.

Another important objective is to ensure secure handling of user input. Web applications receive input through login forms, search boxes, registration forms, and feedback pages. If these inputs are not properly validated, attackers may inject malicious data to exploit the application.

Finally, security testing verifies that data is transmitted and stored securely, helping to maintain user privacy and prevent information leakage.

6.4.5.2 Common Security Vulnerabilities in Web Applications

A vulnerability is a weakness in a software application that can be exploited by attackers. Most successful cyberattacks occur because applications contain one or more

SGOU

vulnerabilities that were not identified during development.

Some of the most common vulnerabilities found in web applications are discussed below.

Authentication Vulnerabilities

Authentication is the process of verifying the identity of a user before granting access to the application. Weak passwords, insecure login mechanisms, or improper password storage can allow attackers to gain unauthorized access.

For example, if an application allows users to create simple passwords such as 123456 or password, attackers can easily guess them using automated tools.

Authorization Vulnerabilities

After users log in successfully, the application should ensure that they can access only the resources permitted for their role. If authorization controls are weak, ordinary users may access confidential information or perform administrative operations.

For instance, if a student can access the administrator's dashboard simply by changing the web address, it indicates an authorization vulnerability.

Input Validation Errors

Almost every web application accepts user input through forms and text fields. If the application fails to validate this input properly, attackers can submit malicious data instead of normal information.

Input validation errors are responsible for many serious attacks, including **SQL** Injection and Cross-Site Scripting (XSS), which will be discussed in the next section.

Session Management Issues

When a user logs in successfully, the application creates a **session** to remember the user's identity while navigating different pages. If session identifiers are predictable or not properly protected, attackers may hijack the session and impersonate legitimate users.

Proper session management ensures that sessions expire after logout or after a period of inactivity.

Sensitive Data Exposure

Many web applications store confidential information such as passwords, credit card numbers, medical records, and personal details. If this information is stored or transmitted without proper encryption, attackers may intercept or steal it.

Using secure communication protocols such as HTTPS and encrypting sensitive data significantly reduces this risk.

SGOU

6.4.5.3 Web Application Security Testing Process

Web application security testing is generally performed in a systematic manner. Although organizations may follow different methodologies, the basic process involves the following stages.

Initially, the tester studies the application to understand its features, user roles, and possible security risks. This is followed by identifying components that require security evaluation, such as login pages, registration forms, search fields, payment modules, and administrative interfaces.

Next, appropriate security testing tools are used to scan the application for known vulnerabilities. Testers may also perform manual testing to identify complex security issues that automated tools might overlook.

After vulnerabilities are identified, each issue is carefully analyzed to determine its severity and possible impact on the application. Finally, the findings are documented in a security testing report, which helps developers understand the identified vulnerabilities and implement suitable corrective measures.

Figure 6.4.1 illustrates the general web application security testing process.

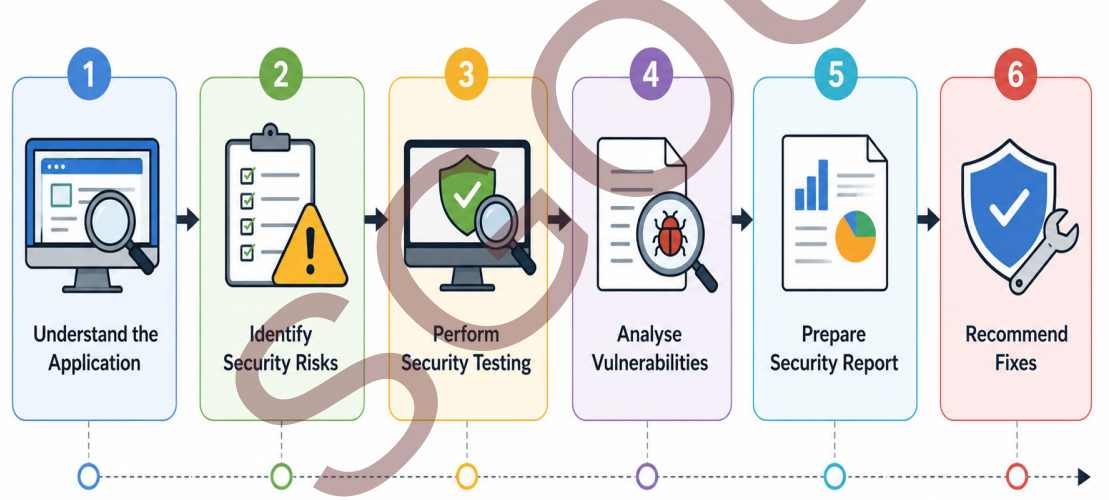


Fig.6.4.1 Web application Security Testing Process

6.4.6 SQL Injection and Cross-Site Scripting (XSS) Testing

In the previous section, you learned that many web application vulnerabilities arise because user inputs are not properly validated. Most web applications accept information from users through login forms, search boxes, registration pages, feedback forms, and comment sections. These inputs are expected to contain valid information such as names, email addresses, passwords, or search keywords.

However, not every user enters genuine information. An attacker may deliberately provide specially crafted input to manipulate the application's behaviour. If the

application processes this input without proper validation, it may perform unintended actions or reveal sensitive information. Such weaknesses are commonly exploited through attacks such as **SQL Injection (SQLi)** and **Cross-Site Scripting (XSS)**.

Security testing helps identify these vulnerabilities before an application is released so that developers can correct them and protect users from potential attacks.

6.4.7 SQL Injection Testing

Most web applications store information such as user accounts, product details, customer records, and transaction histories in a database. Whenever a user logs in or searches for information, the application communicates with the database by sending SQL (Structured Query Language) statements.

A **SQL Injection (SQLi)** attack occurs when an application accepts user input without proper validation and allows it to become part of an SQL query. Instead of processing normal input, the database may interpret malicious input as SQL commands. As a result, attackers may gain unauthorized access to data, modify information, or even delete database records.

Consider a login page where a user enters a username and password. If the application does not validate these inputs properly, an attacker may attempt to manipulate the query sent to the database. Although modern applications use several protection mechanisms, poorly designed applications may still be vulnerable to such attacks.

It is important to understand that SQL Injection is possible because the application trusts user input without verifying whether it contains only valid data. Proper input validation and secure programming practices can effectively prevent this vulnerability.

6.4.7.1 Types of SQL Injection

SQL Injection attacks can occur in different forms depending on how the attacker interacts with the application. The commonly encountered types include:

- **In-band SQL Injection:** The attacker uses the same communication channel to send malicious input and receive the results. This is the most common type.
- **Blind SQL Injection:** The application does not directly display database information. Instead, attackers infer information by observing how the application responds to different inputs.
- **Out-of-band SQL Injection:** Information is retrieved using an alternative communication channel instead of the normal response from the application. This type is less common and depends on the capabilities of the database server.

A successful SQL Injection attack can have serious consequences for an organization.

- Unauthorized access to confidential information
- Theft of customer records and financial data
- Modification or deletion of database contents

SGOU

- Loss of data integrity
- Financial losses and legal consequences
- Damage to an organization's reputation

These consequences demonstrate why database security is a critical aspect of web application development.

6.4.7.2 SQL Injection Testing

SQL Injection testing is performed to determine whether an application properly validates user inputs before sending SQL queries to the database.

During testing, security testers examine input fields such as login forms, search boxes, registration forms, and URL parameters to verify that the application safely processes user input. Automated security testing tools also analyze applications for potential SQL Injection vulnerabilities and generate reports that help developers fix the identified issues.

6.4.7.3 Preventing SQL Injection

Developers can significantly reduce the risk of SQL Injection by following secure programming practices.

- Validate and sanitize all user inputs.
- Use parameterized queries or prepared statements instead of dynamically constructing SQL queries.
- Apply the principle of least privilege to database accounts.
- Display generic error messages instead of detailed database errors.
- Perform regular security testing throughout software development.

Following these practices greatly improves the security of database-driven applications.

6.4.8 Cross-Site Scripting (XSS) Testing

Another common web application vulnerability is **Cross-Site Scripting (XSS)**.

Unlike SQL Injection, which primarily targets the database, XSS targets the users of a web application. It occurs when an application accepts user input and displays it in a web page without proper validation or encoding. As a result, malicious scripts may execute within the user's browser.

Many websites allow users to submit comments, reviews, messages, or profile information. If the application displays this content without proper validation, attackers may attempt to inject malicious scripts into the page. When another user visits the affected page, the browser may execute the script, leading to unintended actions.

SGOU

The primary objective of an XSS attack is not usually to damage the website itself but to compromise the security of its users.

6.4.8.1 Types of Cross-Site Scripting

Cross-Site Scripting attacks are generally classified into three categories.

- **Stored XSS:** The malicious script is permanently stored by the application, for example in a database, and is executed whenever another user views the affected page.
- **Reflected XSS:** The malicious script is included in a request and immediately reflected back by the application in its response.
- **DOM-Based XSS:** The vulnerability exists within the client-side JavaScript code of the web page rather than on the server.

Although these types differ in their execution, they all exploit insufficient validation or encoding of user input.

Cross-Site Scripting attacks can affect both users and organizations.

Possible consequences include:

- Theft of user session information
- Unauthorized access to user accounts
- Display of misleading or altered web content
- Redirection to malicious websites
- Loss of user trust and confidence
- Damage to the organization's reputation

6.4.8.2 XSS Testing

Cross-Site Scripting testing is performed to verify whether an application properly validates and safely displays user-supplied data.

Security testers examine web pages that accept user input, such as registration forms, search pages, feedback forms, discussion forums, and comment sections. They evaluate whether user input is correctly filtered and encoded before being displayed in the browser. Security testing tools can also assist in detecting potential XSS vulnerabilities.

Preventing XSS

The following practices help prevent Cross-Site Scripting vulnerabilities.

- Validate all user inputs.
- Encode output before displaying user-supplied data in web pages.
- Avoid directly inserting user input into HTML pages.

SGOU

- Implement appropriate Content Security Policies (CSP).
- Perform regular security testing and code reviews.

These measures significantly reduce the likelihood of malicious scripts executing in users' browsers.

SQL Injection and XSS: A Comparison

Although SQL Injection and Cross-Site Scripting both result from improper input validation, they differ in their objectives and targets.

Feature	SQL Injection	Cross-Site Scripting (XSS)
Primary Target	Database	User's Web Browser
Main Objective	Manipulate database operations	Execute malicious scripts in the browser
Affects	Database and stored information	Users interacting with the application
Common Cause	Improper validation of database inputs	Improper validation or encoding of displayed content
Prevention	Parameterized queries, input validation	Input validation, output encoding, Content Security Policy

6.4.9 Static Application Security Testing (SAST)

Static Application Security Testing (SAST) is a testing approach in which the application's source code, bytecode, or compiled code is analyzed without executing the application. Since the application is not running during the analysis, SAST is often referred to as white-box security testing because the tester has access to the internal structure of the software.

The primary objective of SAST is to identify security vulnerabilities during the early stages of software development. It examines the code to detect programming errors, insecure coding practices, and potential vulnerabilities before the application is deployed.

For example, suppose a developer accidentally stores passwords in plain text or fails to validate user input before processing it. A SAST tool can identify these issues by analysing the source code and alerting the developer before the application is released.

Since vulnerabilities are detected early, developers can correct them immediately, reducing the cost and effort required for maintenance later.

SGOU

Some common issues detected through SAST include:

- Improper input validation
- Hard-coded passwords or secret keys
- Insecure coding practices
- Buffer overflow risks
- Potential SQL Injection vulnerabilities
- Potential Cross-Site Scripting vulnerabilities

Although SAST is highly effective for analysing source code, it cannot detect vulnerabilities that arise only when the application is running, such as server configuration issues or runtime authentication problems.

Examples of popular SAST tools include SonarQube, Checkmarx, Fortify Static Code Analyzer, and Veracode Static Analysis.

6.4.10 Dynamic Application Security Testing (DAST)

Unlike SAST, Dynamic Application Security Testing (DAST) evaluates an application while it is running. Instead of analysing the source code, DAST interacts with the application in the same way as an external user or attacker would. For this reason, DAST is commonly known as black-box security testing.

During dynamic testing, the application is deployed in a testing environment, and various requests are sent to different web pages, forms, and services. The testing tool analyzes how the application responds and identifies vulnerabilities that may be exploited during actual operation.

For example, a tester may evaluate whether a login page is vulnerable to SQL Injection, whether user input is correctly validated, or whether confidential information is exposed through HTTP responses.

Because DAST analyzes the running application, it can identify security issues that are difficult to detect through source code analysis alone.

Some common vulnerabilities detected through DAST include:

- SQL Injection
- Cross-Site Scripting (XSS)
- Authentication weaknesses
- Session management issues
- Security misconfigurations
- Sensitive data exposure

SGOU

One limitation of DAST is that it cannot identify vulnerabilities in parts of the application that are not executed during testing. It also provides less information about the exact location of vulnerabilities within the source code.

Examples of widely used DAST tools include OWASP ZAP, Burp Suite, Acunetix, and Netsparker.

6.4.11 CSS Validator Tool

A well-designed website is not only secure but also visually appealing, consistent, and easy to use. The appearance of a web page is controlled using Cascading Style Sheets (CSS), which define the colours, fonts, spacing, layouts, and other visual elements of a website. As web applications become larger and more complex, CSS files also grow in size, increasing the possibility of coding errors.

Even small mistakes in CSS, such as a missing semicolon, an incorrect property name, or an invalid value, may cause a web page to display differently across browsers. Such errors affect the usability and user experience of the application. Therefore, developers should verify that their CSS code follows accepted web standards before deploying the website.

A CSS Validator is a software tool that checks whether a CSS file complies with the official CSS standards defined by the World Wide Web Consortium (W3C). It analyzes the CSS code, identifies syntax errors and invalid declarations, and provides feedback that helps developers improve the quality of their style sheets.

Although CSS validation is not intended to identify security vulnerabilities, it contributes to developing reliable, maintainable, and standards-compliant web applications.

6.4.11.1 Working of a CSS Validator

A CSS Validator works by analysing the CSS code submitted by the developer. The code may be provided in different ways, such as by entering the URL of a website, uploading a CSS file, or directly pasting the CSS code into the validator.

The validator compares each CSS rule with the specifications defined in the CSS standard. If it detects any problems, it generates a validation report indicating the location and nature of each issue. Developers can then correct the reported errors and validate the code again until no significant issues remain.

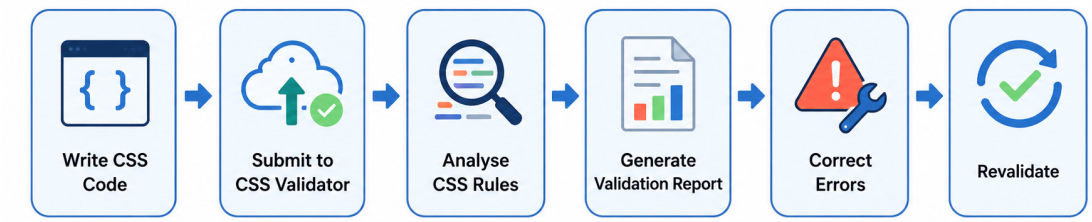


Fig. 6.4.2 Working of CSS Validator

SGOU

6.4.11.2 Understanding Validation Results

After analysing the CSS code, the validator generates a report. Understanding this report helps developers improve the quality of their style sheets.

A validation report generally contains three types of messages.

Errors

Errors indicate violations of CSS syntax or standards that should be corrected. These problems may prevent certain styles from being applied correctly.

Examples include:

- Missing semicolons.
- Incorrect property names.
- Invalid property values.
- Missing braces.

Warnings

Warnings do not necessarily indicate incorrect CSS, but they draw attention to potential issues that developers should review.

For example, a warning may indicate the use of a CSS feature that has limited browser support or a property that is deprecated.

Informational Messages

Some validators also provide informational messages or suggestions to help developers write cleaner and more maintainable CSS code. These messages are advisory and do not indicate errors.

By carefully reviewing these messages, developers can improve both the correctness and quality of their CSS.

6.4.11.3 W3C CSS Validator

One of the most widely used CSS validation tools is the **W3C CSS Validator**, developed by the World Wide Web Consortium (W3C), the organization responsible for developing web standards.

The validator checks CSS code against official specifications and provides a detailed validation report. Developers can validate CSS by:

- Entering the URL of a web page.
- Uploading a CSS file.
- Copying and pasting CSS code directly into the validator.

The validation report highlights errors, warnings, and suggestions, enabling developers to identify and correct problems efficiently.

The W3C CSS Validator is widely used because it is simple, reliable, and based on internationally accepted web standards.

SGOU

Recap

Security Testing and Security Testing Tools

- Protects applications from cyber threats.
- Identifies vulnerabilities before attackers exploit them.
- Based on the CIA Triad: Confidentiality, Integrity, and Availability (CIA).
- Performed throughout the SDLC.

Common Security Risks

- Weak authentication.
- Improper authorization.
- Poor input validation.
- Sensitive data exposure.
- Session management issues.

Security Testing Tools

- Automate vulnerability detection.
- Generate security reports.
- Common tools: OWASP ZAP, Burp Suite, Acunetix, Nessus.

Web Application Security Testing

- Protects web applications from attacks.
- Verifies authentication, authorization, and input validation.
- Identifies vulnerabilities before deployment.

SQL Injection (SQLi)

- Targets databases.
- Caused by improper input validation.
- Prevented using parameterized queries and input validation.

Cross-Site Scripting (XSS)

- Targets users' browsers.
- Executes malicious scripts.
- Prevented by input validation and output encoding.

Static Application Security Testing (SAST)

- Tests source code without execution.
- White-box testing.
- Detects vulnerabilities early.

Dynamic Application Security Testing (DAST)

- Tests the running application.
- Black-box testing.
- Detects runtime vulnerabilities.

SGOU

CSS Validator

- Checks CSS against W3C standards.
- Detects CSS errors and warnings.
- Improves code quality and browser compatibility.

Objective Type Questions

1. What does CIA stand for in information security?
2. Which software testing technique identifies security vulnerabilities?
3. Which principle ensures that only authorized users can access information?
4. Which principle ensures that data is not modified without authorization?
5. Which principle ensures that systems remain accessible to authorized users?
6. Which type of testing focuses on protecting applications from cyber threats?
7. What are weaknesses in software applications called?
8. Which attack targets a database?
9. Which attack targets a user's web browser?
10. What does SQL stand for?
11. Expand XSS.
12. Expand SQLi.
13. Which SQL Injection type uses the same communication channel to send and receive data?
14. Which SQL Injection type does not directly display database information?
15. Which type of XSS permanently stores malicious scripts?
16. Which type of XSS immediately reflects malicious input?
17. Which type of XSS occurs in client-side JavaScript?
18. Which testing approach analyzes source code without executing the application?
19. Which testing approach analyzes a running application?

SGOU

20. What is SAST commonly known as?

21. What is DAST commonly known as?

22. Which security testing tool is developed by OWASP?

23. Which security testing tool intercepts HTTP requests and responses?

24. Which tool automatically scans websites for SQL Injection and XSS?

25. Which tool scans systems and networks for security weaknesses?

26. Which organization developed the CSS Validator?

27. What does W3C stand for?

28. Which validator checks CSS against W3C standards?

29. Which protocol helps protect sensitive data during transmission?

30. What policy helps prevent Cross-Site Scripting attacks?

Answers to Objective Type Questions

1. Confidentiality, Integrity, Availability
2. Security Testing
3. Confidentiality
4. Integrity
5. Availability
6. Security Testing
7. Vulnerabilities
8. SQL Injection (SQLi)
9. Cross-Site Scripting (XSS)
10. Structured Query Language
11. Cross-Site Scripting
12. SQL Injection
13. In-band SQL Injection
14. Blind SQL Injection
15. Stored XSS
16. Reflected XSS

SGOU

17. DOM-Based XSS
18. SAST
19. DAST
20. White-box Testing
21. Black-box Testing
22. OWASP ZAP
23. Burp Suite
24. Acunetix
25. Nessus
26. W3C
27. World Wide Web Consortium
28. W3C CSS Validator
29. HTTPS
30. Content Security Policy (CSP)

Assignments

1. Explain the concept of Security Testing. Discuss its objectives and the importance of the CIA Triad in ensuring application security.

2. Describe the process of Web Application Security Testing. Explain the common security vulnerabilities that may be identified during testing.

3. Explain the concepts of Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). Compare their features, advantages, and limitations.

4. An online shopping website allows users to enter product reviews. A hacker injects a malicious script into the review section, which executes in the browsers of other users viewing the page. Analyze the security vulnerability involved and suggest suitable preventive measures.

5. A banking application directly uses user input to construct SQL queries for database access. During testing, it is found that unauthorized users can manipulate database queries and access confidential customer information. Analyze the vulnerability, discuss its possible consequences, and recommend appropriate methods to prevent such attacks.

SGOU

Suggested Reading

1. OWASP Foundation. (2021). *OWASP Top 10: The Ten Most Critical Web Application Security Risks*. https://owasp.org/www-project-top-ten/

2. World Wide Web Consortium. (n.d.). *W3C CSS Validation Service*. https://jigsaw.w3.org/css-validator/

3. PortSwigger. (2025). *Web Security Academy*. https://portswigger.net/web-security

4. Mozilla. (n.d.). *MDN Web Docs: Web Security*. https://developer.mozilla.org/en-US/docs/Web/Security

5. NIST. (2020). *Security and Privacy Controls for Information Systems and Organizations (SP 800-53 Rev. 5)*. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

Reference

1. Stuttard, D., & Pinto, M. (2021). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley.

2. Kim, D., & Solomon, M. G. (2022). *Fundamentals of Information Systems Security* (4th ed.). Jones & Bartlett Learning.

3. Vacca, J. R. (2020). *Computer and Information Security Handbook* (3rd ed.). Elsevier.

4. Stallings, W., & Brown, L. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson.

5. Goodrich, M. T., & Tamassia, R. (2022). *Introduction to Computer Security* (2nd ed.). Pearson.

SGOU

MODEL QUESTION PAPER SETS

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: ……

Reg. No:.....

Name: ……..

BACHELOR OF COMPUTER APPLICATIONS, SIXTH SEMESTER

END SEMESTER EXAMINATION

Discipline Specific Elective Course

B21CA15DE: Quality Control

2023 -24 Admission Onwards (CBCS -UG)

MODEL QUESTION PAPER - SET I

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. What is the full form of STLC?
2. Which quality attribute protects software from unauthorized access?
3. Which business model encourages community collaboration?
4. Identify the Cost of Quality incurred before product delivery.
5. Which review technique involves systematically checking a work product against requirements, standards, or checklists?
6. Which meeting is used to plan the work of an upcoming Sprint?
7. What is a software quality audit?
8. What does ISO stand for?
9. What is the main purpose of Software Quality Assurance?
10. List any two types of agile methodologies.
11. Write any two limitations of IEEE 1012 standard.
12. Which question addresses the purpose of validation?
13. What is the activity of studying collected data called?
14. Which maturity level of CMMI emphasizes quantitative management?
15. What is a defect tracking tool?

SGOU

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. Why is the Code of Ethics Important?
17. What is a Software Business Model?
18. List the five participants in an Inspection Review and mention their roles.
19. Mention any four objectives of a Project Launch Review.
20. What is a product audit? Give an example.
21. What is CMMI? State its main purpose.
22. Write any four Importance of the V-Model.
23. Define risk exposure and risk rating.
24. Explain how a test management tool supports the software testing process.
25. Differentiate between a mobile device emulator and a simulator.

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. What are the characteristics of good software quality?
27. Discuss the four components of CoQ.
28. Describe the characteristics of an Agile Business Model.
29. Differentiate between internal audits and external audits
30. Explain the major stages involved in the audit process
31. Explain agile software development methodology
32. Explain the life cycle process framework of ISO/IEC/IEEE 12207:2026
33. Explain the CMMI (Capability Maturity Model Integration) framework and its maturity levels.
34. Explain the role of functional testing tools and performance testing tools in software quality assurance.
35. Describe the major steps involved in testing a web application for SQL injection and Cross-Site Scripting (XSS) vulnerabilities.

SGOU

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Explain the Software Quality Plan (SQP). Discuss its objectives, importance, components, and software quality planning process with a suitable example.

37. Explain Formal Review Methods in software quality control.

38. Describe the different types of audits used in software organisations, including internal audits, external audits, product audits, process audits, system audits, compliance audits, and security audits.

39. Explain the Practical Software and Systems Measurement (PSM) method, its key steps, and its importance in software project management.

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: ……

Reg. No:.....

Name: ……..

BACHELOR OF COMPUTER APPLICATIONS, SIXTH SEMESTER

END SEMESTER EXAMINATION

Discipline Specific Elective Course

B21CA15DE: Quality Control

2023 -24 Admission Onwards (CBCS -UG)

MODEL QUESTION PAPER - SET II

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. Which report summarises the overall testing activities?
2. Which quality attribute makes software easy to modify?
3. Which project dimension represents the project budget?
4. Name the standard used for Environmental Management.
5. Which quality model organises software quality factors into Product Operation, Product Revision, and Product Transition?
6. Name the process used to evaluate project progress and performance.
7. What does SCAMPI stand for?
8. Which SQA activity examines software work products to identify errors at an early stage?
9. Which framework is used by SCAMPI to assess software development processes?
10. What question does verification seek to answer?
11. Write any two principles of requirements validation.
12. What is the purpose of ISO/IEC/IEEE 12207.
13. Which ISO standard focuses on customer satisfaction?
14. Which CMMI process area is responsible for software measurement?
15. What is Selenium?

SGOU

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. What is Software Ethics?
17. Compare Quality Assurance (QA) and Quality Control (QC).
18. Differentiate between a Quality Standard and a Quality Model with suitable examples.
19. Mention any four objectives of a Project Launch Review.
20. What is a process audit? State its purpose.
21. What is a software review? State its purpose
22. Write the purpose and scope of the IEEE 1012 standard.
23. Define risk probability and risk impact.
24. Differentiate between static security testing and dynamic security testing
25. Explain how cross-browser testing helps ensure the reliability of a web application.

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. Explain the major quality attributes of software.
27. Discuss the differences between Personal Review and Desk-Check Review
28. Describe the major characteristics of the ISO/IEC 25010 software quality model.
29. Explain the major stages involved in the audit process
30. Describe the five maturity levels of the CMMI maturity model
31. Explain the V-Model.
32. Explain the testing done in the validation Phase.
33. Explain the different roles and responsibilities involved in risk management.
34. Explain the role of Selenium in automated functional testing and describe its major benefits.
35. Describe the importance of mobile compatibility testing and mobile app performance testing in developing reliable mobile applications.

SGOU

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Explain the five dimensions of a software project. Discuss the importance of scope, time, cost, quality, and resources.

37. Define Software Review and explain its major objectives. Discuss the different types of software reviews with suitable examples.

38. Explain the Capability Maturity Model Integration (CMMI) and discuss its maturity levels and importance in software process improvement.

39. Explain the major automated testing tools, including Selenium, cross-browser testing tools, load testing tools, and defect tracking tools.

SGOU

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ്വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kollam
Sreenarayanaguru Open University
Headquarters Building
BSNL Bhavan, Vellayittambalam,
Kollam, Kerala, Pin : 691012
Ph: 9188912554
email:rckollam@sgou.ac.in

Tripunithura
Govt. College
Tirpunithura, Ernakulam
Kerala, Pin : 682301
Ph: 04842927436
email:rcedirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery
Kannur, Pin : 670106
Ph: 04952990494
email:rctdirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad
Kerala, Pin : 679303
Ph: 04662912009
email:rcpdirector@sgou.ac.in

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin : 673002
Ph: 04952920228
email:rckdirector@sgou.ac.in

SAY NO TO RAGGING

BEFORE YOU EVEN THINK OF RAGGING



Humiliation

Blacklisting

Ruined Career

Download

ANTI RAGGING

App

Don't just stand and watch. Stop Ragging! Show Character

Visit UGC Website i.e. www.ugc.ac.in & www.antiraging.in to UGC Anti Ragging regulations.

Ministry of Education
Government of India

Are You Being Ragged ?

विश्वविद्यालय अनुदान आयोग
University Grants Commission

SSGOU

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY



SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

Quality Control

COURSE CODE: B21CA15DE

SGOU

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841

ISBN 978-81-903304-8-0