Software Quality Assurance and Testing

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

WORK INTEGRATED LEARNING PROGRAMMES

COURSE HANDOUT

Part A: Content Design

Course Title	Software Quality Assurance and Testing
Course No(s)	SE ZG501
Credit Units	4
Course Author	Bhaskar Zeminder
Version No	1.0
Instructor –in-Charge	Rajesh S
Date	29- July-2023

Course Objectives

No	Objective
CO1	Develop a comprehensive understanding of software quality assurance principles, methodologies, and industry best practices to ensure the delivery of high-quality software products
CO2	Master the various techniques and tools used in software testing, verification, and validation to effectively identify and rectify defects throughout the software development lifecycle.
CO3	Analyse and apply advanced quality assurance strategies such as test automation, continuous integration, and agile testing, to enhance the efficiency and effectiveness of software development teams in delivering top-notch software products

Textbook(s)

T1	Software Quality Assurance Book by Alain April and Claude Y. Laporte
T2	Software Quality Assurance (From Theory to Implementation) by Daniel Galin

Reference Book(s) & other resources

R1	Software Quality Assurance By Ivan Mistrik, Richard M Soley, Nour Ali, John Grundy, Bedir Tekinerdogan
R2	Software Testing: Concepts and Operations by Rajiv Chopra
R3	Software Testing and Quality Assurance – Theory and Practice, Kshirasagar Naik, Priyadarshi Tripathy, Wiley, 2013
R4	Software Quality Engineering – Jeff Tian, Wiley India, 2015
R5	Quality Planning and Assurance Book by Herman Tang

Content Structure

Module 1: Essential SQA: Processes and Success Factors

Topic No.	Topic Title	Reference
1.1	Definition and importance of software quality assurance	T1 Chapter 1
1.2	Distinction between Quality Assurance and Quality Control	Lecture Notes
1.3	Success Factors in Quality Assurance	T1 Chapter 1 & T2 Chapter 1
1.4	Cost of Quality and Quality Culture	T1 Chapter 2
1.5	Role of SQA in software development life cycle	Lecture Notes

Module 2: Standardizing SQA: Quality Models and Management

Topic No.	Topic Title	Reference
2.1	Software Quality Models	T1 Chapter 3, R1 Chapter 2
2.2	Specifying Quality Requirements and Plan	T1 Chapter 3
2.3	Requirement Traceability During Software Lifecycle	T1 Chapter 3
2.4	Standards for Quality Management	T1 Chapter 4
2.5	Frameworks (ITIL, ISO, CMMi)	T1 Chapter 4

Module 3: Fundamentals of SQA: Software Quality Attributes

Topic No.	Topic Title	Reference
3.1	Software Requirements into Software Quality Factors	T2 Chapter 3
3.2	Understanding quality attributes • Reliability • Usability • Maintainability • Other quality attributes	T2 Chapter 3, R1 Chapter 2
3.3	Alternative models of Software Quality Factors	T2 Chapter 3

Module 4: Deep driving SQA: Software Testing Techniques

Topic No.	Topic Title	Reference
4.1	Software Testing Fundamentals	T2 Chapter 9, R2 Chapter 1
4.2	Software Verification and Validation	R2 Chapter 2
4.3	Test design techniques (black-box testing, white-box testing, boundary value analysis, equivalence partitioning, etc.)	R2 Chapter 3 & 4
4.4	Test levels and types (unit testing, integration testing, system testing, etc.)	R2 Chapter 7

Module 5: Mastering SQA: Test Execution and Automated Testing

Topic No.	Topic Title	Reference
5.1	Test Execution Process	T2 Chapter 10
5.2	Test Case Design	T2 Chapter 10
5.3	Automated testing	T2 Chapter 10, R2 Chapter 9

5.4	Alpha and Beta site testing programs	T2 Chapter 10
5.5	Regression Testing Strategies	R2 Chapter 6, R2 Chapter 12
5.6	Case Study: Exploring Automated Source Code Analyzers and Software Composition Analysis Tools	Lecture Notes

Module 6: Effective SQA: Quality Audits and Project Assessments

Topic No.	Topic Title	Reference
6.1	Personal Review, Inspection Review and Project Assessments	T1 Chapter 5, T2 Chapter 8
6.2	Types of Audits (Internal, Third Party)	T1 Chapter 6
6.3	Project Assessment and Control Process	T1 Chapter 6, T2 Chapter 8
6.4	Corrective Actions	T1 Chapter 8

Module 7: Comprehensive SQA: Effective Test Management and Planning

Topic No.	Topic Title	Reference
7.1	Test Organization and Team Management	T1 Chapter 5
7.2	Test Estimation and Scheduling	R1 Chapter 7
7.3	Test Data Management	Lecture Notes
7.4	Configuration Management and Change Control	T1 Chapter 8, T1 Chapter 5
7.5	Case Study: Develop a test plan and design test cases for a given software application	Lecture Notes

Module 8: Enhancing SQA: Process Improvement and Metrics

Topic No.	Topic Title	Reference

8.1	Introduction to Test Process Improvement	T1 Chapter 9
8.2	Capability Maturity Model Integration (CMMI) for Testing	T1 Chapter 10
8.3	Six Sigma in Software Testing	Lecture Notes
8.4	Test Metrics for Process Improvement	T2 Chapter 21

Module 9: Optimizing SQA: Agile Testing and DevOps Integration

Topic No.	Topic Title	Reference
9.1	Introduction to Agile Methodology and Testing	T1 Chapter 4
9.2	Agile Test Planning and Execution	T1 Chapter 5
9.3	Continuous Testing in DevOps	Lecture Notes
9.4	Test Environment and Test Data Management in DevOps	Lecture Notes

Module 10: Excelling SQA: Best Practices and Case Studies

Topic No.	Topic Title	Reference
10.1	Best Practices for SQA implementation	T1 Chapter 4, Lecture Notes
10.2	Quality assurance in different development methodologies (Waterfall, Agile, etc.)	T2 Chapter 7
10.3	Building a quality culture in organizations	R1 Chapter 7
10.4	Case studies of successful SQA implementations	Lecture Notes
10.5	Lessons learned from successful software quality assurance projects	Lecture Notes

Module 11: Shaping SQA: An Outlook for the Future

Topic No.	Topic Title	Reference
11.1	Emerging technologies and their impact on SQA	Lecture Notes
11.2	Artificial intelligence and machine learning in quality assurance	Lecture Notes
11.3	Blockchain and quality assurance	Lecture Notes
11.4	Future directions and career opportunities in SQA	Lecture Notes

Learning Outcomes:

No	Learning Outcomes	Objectives
LO1	Advanced knowledge of Software Quality Assurance principles and methodologies	By the end of the course, participants should have an in-depth understanding of the advanced principles, concepts, and methodologies of Software Quality Assurance. They should be well-versed in topics like quality models, process improvement frameworks (e.g., CMMI, Six Sigma), risk management, and the application of SQA in different software development models (e.g., Agile, DevOps).
LO2	Master testing techniques and tools for comprehensive software testing.	Upon completing the course, participants should have a solid knowledge of various testing techniques, such as unit testing, integration testing, system testing, and acceptance testing. They should also be proficient in using popular testing tools and frameworks to create, execute, and analyse test cases effectively, ensuring comprehensive test coverage and defect detection.
LO3	Proficiency in designing and implementing robust quality assurance strategies	Upon completing the course, participants should be capable of developing comprehensive and effective quality assurance strategies tailored to specific software projects. They should be skilled in devising test plans, defining test cases, and establishing quality metrics to measure the performance and reliability of software applications across different domains and industries.

Part B: Contact Session Plan

Academic Term	First Semester 2024-2025

Course Title	Software Quality Assurance and Testing
Course No	SE ZG501
Lead Instructor	Rajesh S

Teaching Methodology *(Online Session Mode)*

The pedagogy for this course is centred around online contact sessions, which consist of 2-hour lecture sessions. In addition to the delivery of lessons on the topics, these contact sessions will also be enriched with discussions on organization-specific practices and case studies from experienced QA managers in the Indian IT industry.

Course Delivery

- There are 16 Contact Sessions (of 2 hours each)—8 before mid-semester and 8 post-mid-semester over a period of 16 weeks
- The 8th & 16th Contact Sessions are planned for review of topics pre-mid-semester and pre-comprehensive examinations.

Course Contents

Contact Session	List of Topic Title
1	Module 1: Essential SQA: Processes and Success Factors
2	Module 2: Standardising SQA: Quality Models and Management
3-4	Module 3: Fundamentals of SQA: Software Quality Attributes
5	Module 4: Deep driving SQA: Software Testing Techniques
6-7	Module 5: Mastering SQA: Test Execution and Automated Testing
8	**Review of Contact Session Topics (1 to 7) for Mid-Sem Examination**
9	Module 6: Effective SQA: Quality Audits and Project Assessments
10-11	Module 7: Comprehensive SQA: Effective Test Management and Planning
12	Module 8: Enhancing SQA: Process Improvement and Metrics
13	Module 9: Optimizing SQA: Agile Testing and DevOps Integration
14	Module 10: Excelling SQA: Best Practices and Case Studies
15	Module 11: Shaping SQA: An Outlook for the Future
16	**Review of All Topics for Comprehensive Examination**

Assignments

Each participant or Group of participants will be given an assignment on a topic that was discussed in class. The assignment topics will be based on practical problems experienced or part of work items or tools used by collaborating organizations.

- Assignments will be take-home and deadline-driven, typically lasting 2 weeks. Participants are expected to spend at least 16 hours on the study, research, discussion, and preparation of the report and presentation.
- As part of the deliverables, participants will prepare a report and/or make a short presentation in class.

Experiential Learning Components

No	Topic	Objectives	Hands-On-Exercises
1	SQA Planning and Design	Learn how to create a comprehensive test plan and design effective test cases.	Develop a test plan and design test cases for a given software application.
2	Automation Frameworks	Introduction to automation tools and frameworks	Automate test cases using a popular test automation tool.
3	Usability Testing	Understand the importance of user experience and usability in software.	Conduct usability testing on a software interface and provide recommendations for improvement.
4	Continuous Integration and Continuous Testing	Explore the concepts of continuous integration and continuous testing in SQA	Demonstration of CI/CD pipeline and automate the testing process for a software application.

Evaluation Scheme:

Legend: EC = Evaluation Component; AN = After Noon Session; FN = Fore Noon Session

Evaluation Component	Name	Type (Open book, Closed	Weight	Duration	Day, Date, Session, Time

	(Quiz, Lab, Project, Mid-term exam, End semester exam, etc.)	book, Online, etc.)			
EC - 1	Quiz-I/ Assignment-I	Online	7.5%		September 1-10, 2024
	Quiz-II	Online	7.5%		October 10-20, 2024
	Quiz-III/ Assignment-II	Online	15%		November 1-10, 2024
EC - 2	Mid-sem	Closed book	30%	2 hours	Friday, 20/09/2024 (AN)
EC - 3	Comprehensive	Open book	40%	2 ½ hours	Friday, 29/11/2024 (AN)

Note: If Assignment kindly remove Quiz-I, II, III

Syllabus for Mid-Semester Test (Closed Book): Topics in Contact Hours: 1 to 8

Syllabus for Comprehensive Exam (Open Book): All topics

Important links and information:

Elearn portal: https://elearn.bits-pilani.ac.in

Students are expected to visit the Elearn portal on a regular basis and stay up to date with the latest announcements and deadlines.

Contact sessions: Students should attend the online lectures as per the schedule provided on the Elearn portal.

Evaluation Guidelines:

1. EC-1 consists of either two Assignments or three Quizzes. Students will attempt them through the course pages on the Elearn portal. Announcements will be made on the portal, in a timely manner.

2. For Closed Book tests: No books or reference material of any kind will be permitted.

3. For Open Book exams: Use of books and any printed / written reference material (filed or bound) is permitted. However, loose sheets of paper will not be allowed. Use of calculators is permitted in all exams. Laptops/Mobiles of any kind are not allowed. Exchange of any material is not allowed.

4. If a student is unable to appear for the Regular Test/Exam due to genuine exigencies, the student should follow the procedure to apply for the Make-Up Test/Exam which will be made available on the Elearn portal. The Make-Up Test/Exam will be conducted only at selected exam centres on the dates to be announced later.

It shall be the responsibility of the individual student to be regular in maintaining the self-study schedule as given in the course handout, attend the online lectures, and take all the prescribed evaluation components such as Assignment/Quiz, Mid-Semester Test and Comprehensive Exam according to the evaluation scheme provided in the handout.

Birla Institute of Technology & Science, Pilani

Work Integrated Learning Programmes Division

Second Semester 2023-2024

Mid-Semester Test

(EC-2 Regular)

Course No.	: SE ZG501
Course Title	: Software Quality Assurance and Testing
Nature of Exam	: Closed Book
Weightage	: 30%
Duration	: 2 Hours
Date of Exam	: 15/03/2024 (AN)

No. of Pages = 1

Note to Students:

1. Please follow all the *Instructions to Candidates* given on the cover page of the answer book.
2. All parts of a question should be answered consecutively. Each answer should start from a fresh page.
3. Assumptions made if any, should be stated clearly at the beginning of your answer.

Q.1 Question – Explain what you understand by Quality Assurance and the term Software Testing. Give five differences between quality assurance and S/W Testing. **Marks - 6**

Q.2 Give the five levels of CMMI-Dev and explain key characteristics of each level. Give two process areas for each level. **Marks - 6**

Q.3 What are the sub characteristics of Security quality requirement in ISO 25010. List them all and explain each in 2-3 lines. Marks - 6

Q.4 What do you understand by Verification and Validation in S/W Testing. Give differences and examples. Marks - 6

Q.5 Given a function that takes input – Month, Day and Year. Also, test designer has formed equivalence classes for the three inputs as

Month – {1,2,3,4,5,6} and {7,8,9,10,11,12)

Day – {1-15], {16-31}

Year – {2001-2024}

Give a set of Weak Normal and Strong Normal Test cases using Equivalence Class Testing methodology. Give reason for your answer. **Marks 6**

******

Birla Institute of Technology & Science, Pilani

Work Integrated Learning Programmes Division

Second Semester 2023-2024

Comprehensive Examination

(EC-3 Regular)

Course No.	: SE ZG501
Course Title	: Software Quality Assurance and Testing
Nature of Exam	: Open Book
Weightage	: 40%
Duration	: 2 ½ Hours
Date of Exam	: 17/05/2024 (AN)

No. of Pages = 2

Note to Students:

4. Please follow all the *Instructions to Candidates* given on the cover page of the answer book.
5. All parts of a question should be answered consecutively. Each answer should start from a fresh page.
6. Assumptions made if any, should be stated clearly at the beginning of your answer.

Q.6 Explain the similarities and differences between Walkthrough, Inspection and Audits. Explain in detail how each of these work. **Marks - 8**

Q.7 What do you mean by Baseline. What is the significance of Baseline. When can a Baseline be created. How can Baseline be changed. **Marks - 6**

Q.8 Give different branching strategies for Source code. Explain how these work, giving examples. In which scenario these strategies work well.

Marks – 6

Q.9 What do you mean by Alpha Testing, Beta Testing and Regression Testing. How can one choose Test Cases in Regression Test Pass. Explain in detail. (Give scenarios/examples when regression testing will be needed and how test cases can be chosen efficiently in each case)

Marks – 6

Q.10 A product was tested for a period of 5 months (Assume all months have 30 Days). The total number of defects found were 120. Assume that product is live for 24 Hrs a day. On an average the time required to fix the defect was 20 Hrs. Find availability of the Product. What can be done to improve the availability by 20%. Assume that incoming bug rate cannot be changed since product is already delivered.

Marks – 6

Q.11 In a train reservation system, the coach is given by two characters. A text box can take two characters as input such that

a. First Character can be F or S
b. Second character can be M or W
c. If First character is A, message displayed “First AC”.
d. If First character is B, message displayed “Second AC”.
e. If First character is not A or B, message displayed “General Coach”
f. If Second character is M, message displayed “Men”.
g. If Second character is W, message displayed “Women”.
h. If Second character is not M or W, message displayed “Both”.
i. Final Message will be combination of message based on First and Second character (First message followed by Second.

Use Decision Table technique so as to find the Test Cases for this problem. Give the message displayed for each Test Case.

Marks – 8

Contents

1. CS1

Topic 1.1: Software Quality Assurance

- SQA Addresses the global challenge of the improvement of software quality.
- It seeks to provide an overview of software quality assurance (SQA) practices for customers, managers, auditors, suppliers, and personnel responsible for software projects, development, maintenance, and software services.
- In a globally competitive environment, clients and competitors exert a great deal of pressure on organizations.
- Clients are increasingly demanding and require, among
- other things, software that is of high quality, low cost, delivered quickly, and with impeccable after-sales support.
- To meet the demand, quality, and deadlines, the organization must use efficient quality assurance practices for their software activities.
- Standards define ways to maximize performance but managers and employees are largely left to themselves to decide how to practically improve the situation.
- They face several problems:
  - **increasing pressure to deliver quality products quickly**
  - **increasing size and complexity of software and of systems**
  - **increasing requirements to meet national, international, and professional standards**
  - **subcontracting and outsourcing**
  - **distributed work teams**
  - **ever changing platforms and technologies.**

Topic 1.2: INTRODUCTION

- Software is developed, maintained, and used by people in a wide variety of situations.
- Students ,enthusiasts , professionals address quality
- problems that arise in the software they are working with
- The Guide to the Software Engineering Body of Knowledge (SWEBOK) [SWE 14] constitutes the first international consensus developed on the fundamental knowledge required by all software engineers.

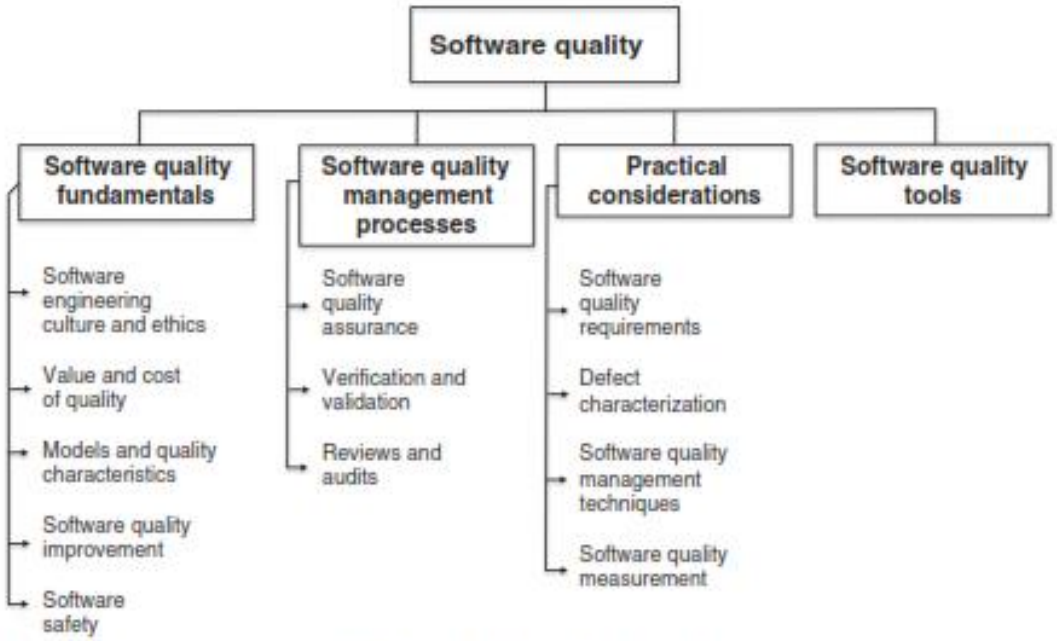


Figure 1.1 Software Quality in the SWEBOK® Guide [SWE 14].

Topic 1.3: DEFINING SOFTWARE QUALITY

- "software," "software quality," and "software quality assurance"
- **Software [** ISO 24765 [ISO 17a]]

1) All or part of the programs, procedures, rules, and associated documentation of an information processing system.

2) Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

- Software found in embedded systems is sometimes called microcode or firmware.
- Firmware is present in commercial mass-market products and controls machines and devices used in our daily lives.
- **Firmware** Combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device.

Topic 1.4: SOFTWARE ERRORS, DEFECTS, AND FAILURES

- The system crashed during production.
- The designer made an error.
- After a review, we found a defect in the test plan.
- I found a bug in a program today.
- The system broke down.
- The client complained about a problem with a calculation in the payment report.
- A failure was reported in the monitoring subsystem.

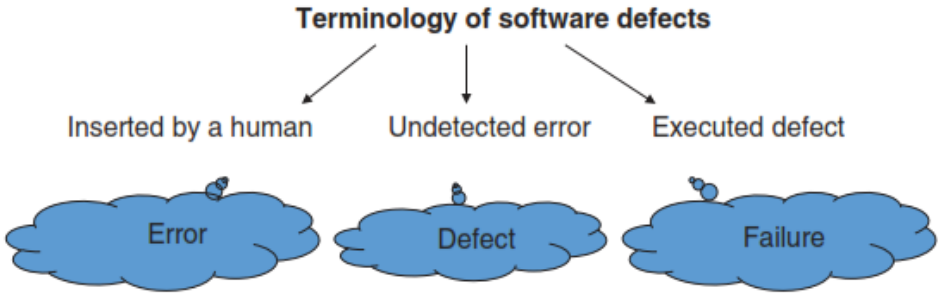
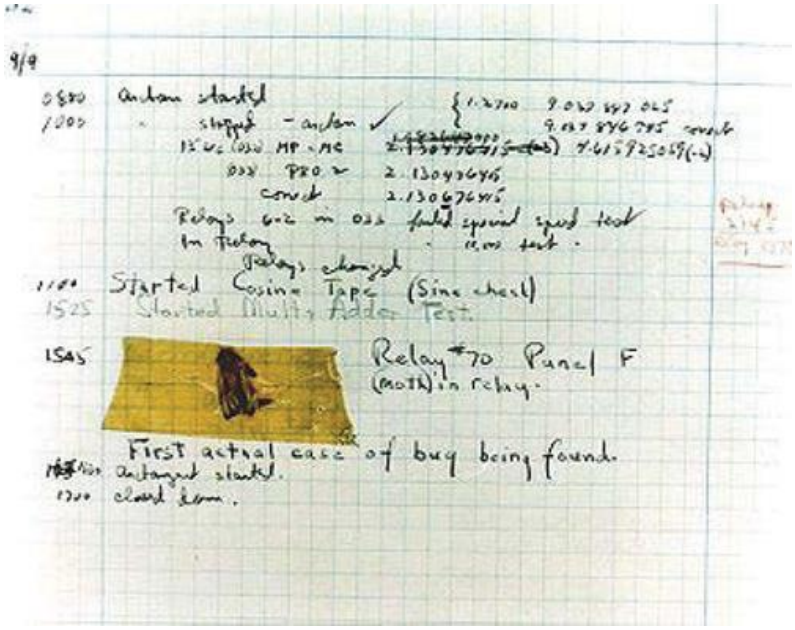
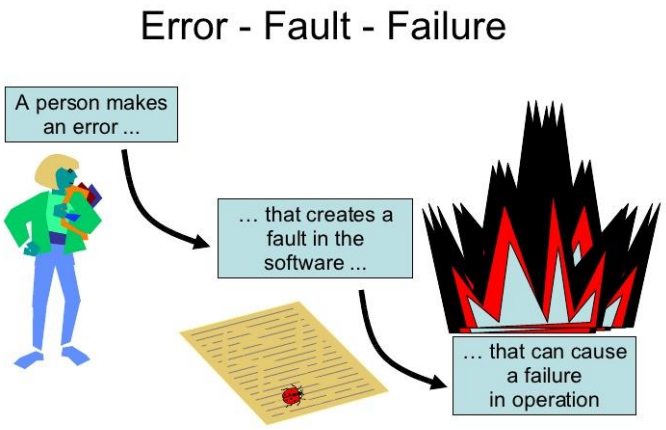


Figure 1.2 Terminology recommended for describing software problems.



- A failure (synonymous with a crash or breakdown) is the execution (or manifestation) of a fault in the operating environment.
- A failure is defined as the termination of the ability of a component to fully or partially perform a function that it was designed to carry out.
- The origin of a failure lies with a defect hidden, that is,
- not detected by tests or reviews, in the system currently in operation.
- Defects (synonym of faults) are human errors that were not detected during software development, quality assurance (QA), or testing.
- An error can be found in the documentation, the software source code instructions, the logical execution of the code, or anywhere else in the life cycle of the system.



- **Error**
  - A human action that produces an incorrect result.
  - (ISO 24765) [ISO 17a]
- **Defect**
  - A problem (synonym of fault) which, if not corrected, could cause an application to either fail or to produce incorrect results.
    - (ISO 24765) [ISO 17a]
  - An imperfection or deficiency in a software or system component that can result in the component not performing its function. For example, an incorrect data definition or source code instruction.
    - A defect, if executed, can cause the failure of a software or system component.
    - (ISTQB 2011) [IST 11i]
- **Failure**
  - The termination of the ability of a product to perform a required function or its inability to perform within previously specified limits.
  - (ISO 25010) [ISO 11i]

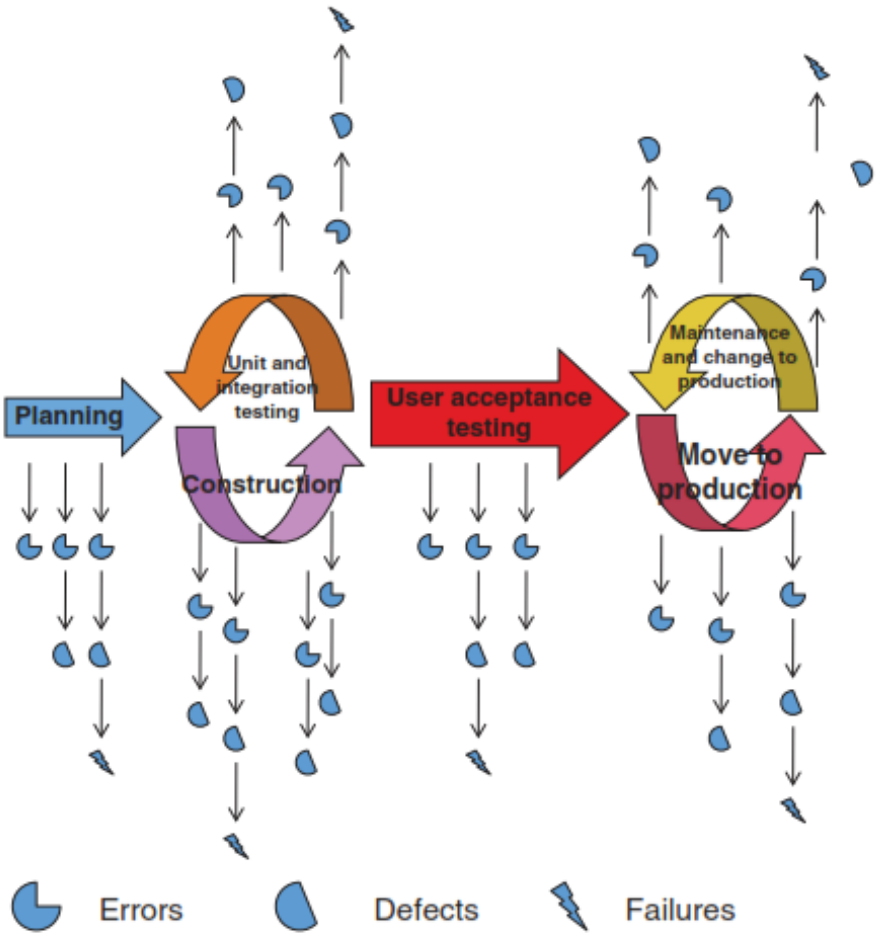


Figure 1.3 Errors, defects, and failures in the software life cycle.

Topic 1.5: Case 1:

A local pharmacy added a software requirement to its cash register to prevent sales of more than $75 to customers owing more than $200 on their pharmacy credit card.

The programmer did not fully understand the specification and created a sales limit of $500 within the program. This defect never caused a failure since no client could purchase more than $500 worth of items given that the pharmacy credit card had a limit of $400.

Topic 1.6: Case 2:

In 2009, a loyalty program was introduced to the clients of American Signature, a large furniture supplier. The specifications described the following business rules: a customer who makes a monthly purchase that is higher than the average amount of monthly purchases for all customers will be considered a Preferred Customer.

The Preferred Customer will be identified when making a purchase, and will be immediately given a gift or major discount once a month.

The defect introduced into the system (due to a poor understanding of the algorithm to set up for this requirement) involved only taking into account the average amount of current purchases and not the customer's monthly history. At the time of the software failure, the cash register was identifying far too many Preferred Clients, resulting in a loss for the company.

Topic 1.7: Development Life Cycle

- Software life cycle process that contains the activities of requirements analysis, design, coding, integration, testing, installation, and support for acceptance of software products.
- Depending on the business model of your organization, you will have to allow for varying degrees of effort in identifying and correcting defects.
- Airbus, Boeing, Bombardier, and Embraer to have
- identified and corrected all the defects in the software for their airplanes before we board them!

Topic 1.8: Research Studies have come to the following conclusions:

- The scope of most defects is very limited and easy to correct.
- Many defects occur outside of the coding activity (e.g., requirement, architecture activities).
- Poor understanding of the design is a recurrent problem in programming error studies.
- It is a good idea to measure the number and origin of defects in your organization to set targets for improvement.

Topic 1.9: SQA need to develop a classification of the causes of software error by category.

1) problems with defining requirements;

2) maintaining effective communication between client and developer;

3) deviations from specifications;

4) architecture and design errors;

5) coding errors (including test code);

6) non-compliance with current processes/procedures;

7) inadequate reviews and tests;

8) documentation errors.

Topic 1.10: SOFTWARE QUALITY

- *Conformance to established software requirements; the capability of a software product to satisfy stated and implied needs when used under specified conditions.*
- *The degree to which a software product meets established requirements; however, quality depends upon the degree to which those established requirements accurately represent stakeholder needs, wants, and expectations*

Topic 1.11: SOFTWARE QUALITY ASSURANCE

- **Software Engineering**

The systematic application of scientific and technological knowledge, methods, and experience for the design, implementation, testing, and documentation of software.

- **Quality Assurance**
  - **1) a planned and systematic pattern of all actions necessary to provide adequate confi**dence that an item or product conforms to established technical requirements;
  - **2) a set of activities designed to evaluate the process by which products are developed or** manufactured;

- **3) the planned and systematic activities implemented within the quality system, and** demonstrated as needed, to provide adequate confidence that an entity will fulfil requirements for quality.

Topic 1.12: Software Quality Assurance Definition

A set of activities that define and assess the adequacy of software processes to provide evidence that establishes confidence that the software processes are appropriate for and produce software products of suitable quality for their intended purposes.

A key attribute of SQA is the objectivity of the SQA function with respect to the project.

The SQA function may also be organizationally independent of the project; that is, free from technical, managerial, and financial pressures from the project.

Topic 1.13: SQA Elements

- The need to plan the quality aspects of a product or service;
- Systematic activities that tell us, throughout the software life cycle, that certain corrections are required;
- The quality system is a complete system that must, in the context of quality management, allow for the setting up of a quality policy and continuous improvement;
- QA techniques that demonstrate the level of quality reached so as to instill confidence in users; and lastly,
- Demonstrate that the quality requirements defined for the project, for the change or by the software department have been met.

Topic 1.14: BUSINESS MODELS AND THE CHOICE OF SOFTWARE ENGINEERING PRACTICES

- **Business Model**
  - A business model describes the rationale of how an organization creates, delivers, and captures value (economic, social, or other forms of value).
  - The essence of a business model is that it defines the manner by which the business enterprise delivers value to customers, entices customers to pay for value, and converts those payments to profit.
- **Choice of Software Practices**
  - As expected, people from different business sectors chose software engineering practices that would lower the probability of their worst fears(quality and deadlines).
  - Since their apprehensions are different, their practices are also different.

Topic 1.15: SUCCESS FACTORS

- **Foster Software Quality**

1) SQA techniques adapted to the environment.

2) Clear terminology with regards to software problems.

3) An understanding and specific attention to each major category of software error sources.

4) An awareness of the SQA body of knowledge of the SWEBOK as a guide for SQA.

- **Factors that may Adversely Affect Software Quality**

1) A lack of cohesion between SQA techniques and environmental factors in your organization.

2) Confusing terminology used to describe software problems.

3) A lack of understanding or interest for collecting information on software error sources.

4) Poor understanding of software quality fundamentals.

5) Ignorance or non-adherence with published SQA techniques.

Topic 1.16: Software quality assurance vs. software quality control

- **Quality control is defined as "a set of activities designed to evaluate the** quality of a developed or manufactured product"
- The main objective of **quality assurance is to minimize the cost of guaranteeing** quality by a variety of activities performed throughout the development and manufacturing processes/stages.
- These activities prevent the causes of errors, and detect and correct them early in the development process.
- Quality assurance activities substantially reduce the rate of products that do not qualify for shipment and, at the same time, reduce the costs of guaranteeing quality in most cases.

Topic 1.17: The objectives of SQA activities

- ***Software development (process-oriented):***
- ***Software maintenance (product-oriented):***

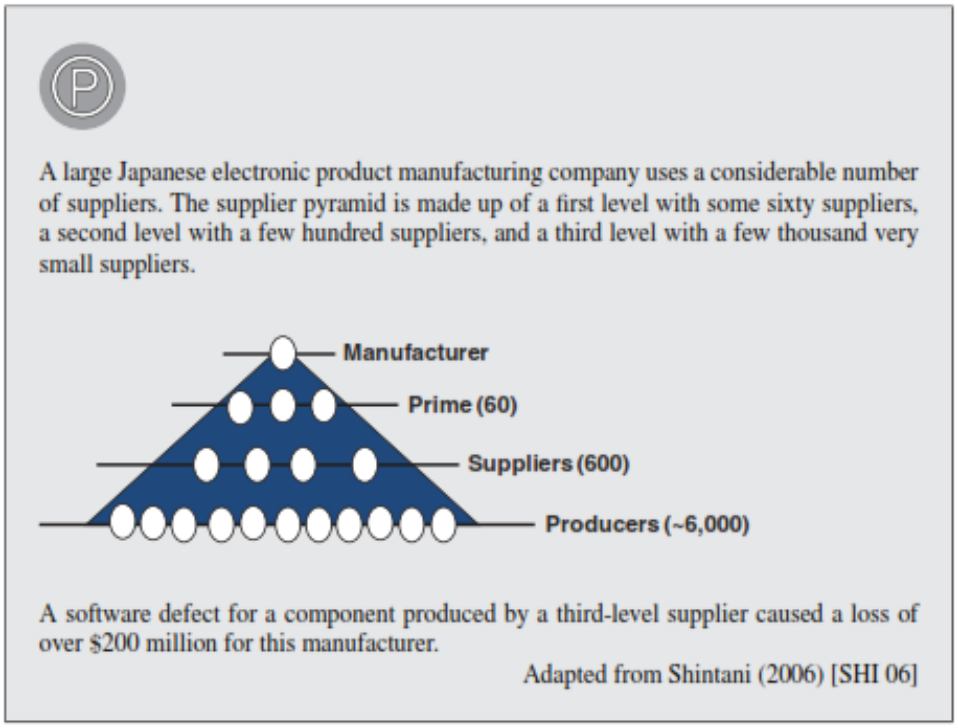
Topic 1.18: Software development (process-oriented):

- 1. Assuring an acceptable level of confidence that the software will conform to functional technical requirements.
- 2. Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements.
- 3. Initiating and managing of activities for the improvement and greater efficiency of software development and SQA activities. This means improving the prospects that the functional and managerial requirements will be achieved while reducing the costs of carrying out the software development and SQA activities.

Topic 1.19: Software maintenance (product-oriented):

- **1. Assuring with an acceptable level of confidence that the software maintenance activities will conform to the functional technical requirements.**
- **2. Assuring with an acceptable level of confidence that the software maintenance activities will conform to managerial scheduling and budgetary requirements.**
- **3. Initiating and managing activities to improve and increase the efficiency of software maintenance and SQA activities. This involves improving the prospects of achieving functional and managerial requirements while reducing costs.**

Topic 1.20: Quality Culture



A large Japanese electronic product manufacturing company uses a considerable number of suppliers. The supplier pyramid is made up of a first level with some sixty suppliers, a second level with a few hundred suppliers, and a third level with a few thousand very small suppliers.

A software defect for a component produced by a third-level supplier caused a loss of over $200 million for this manufacturer.

Adapted from Shintani (2006) [SHI 06]

- Setting up a quality culture and software quality assurance (SQA) principles, as stipulated in the standards, could help solve these problems.
- Quality, influenced by organization's senior management
- and the organization's culture, has a cost, has a positive effect on profits, and must be governed by a code of ethics.

Topic 1.21: COST OF QUALITY

- One of the major factors that explains the resistance to implementing quality assurance is the perception of its high cost.
- As software engineers, responsible for informing administrators of the risks that a company takes when not fully committed to the quality of its software.
- A practical manner of beginning the exercise is to identify the costs of non-quality.
- It is easier to identify potential savings by studying the problems caused by software.

Topic 1.22: Costs of a project

(1) Implementation costs

(2) Prevention costs

(3) Appraisal costs

(4) The costs associated with failures or anomalies.

- If all development activities are error free, then 100% of costs could be implementation costs.
- Given that we make mistakes, we need to be able to identify them. The costs of detecting errors are appraisal costs (e.g., testing).
- Costs due to errors are anomaly costs.

- When we want to reduce the cost of anomalies, we invest in training, tools, and methodology. These are prevention costs.

➢ The "cost of quality" is not calculated in the same way in all organizations.

- There is a certain amount of ambiguity between the notion of the cost of quality, the cost of non-quality, and the cost of obtaining quality.
- Most commonly used model at this time, costs of quality take into account the following five perspectives:

(1) prevention costs, (2) appraisal costs, (3–4) failure costs (internal during development, external on the client's premises), and (5) costs associated with warranty claims and loss of reputation caused by non-quality.

Topic 1.23: calculation of the cost of quality in this model is as follows:

Quality costs = Prevention costs

+ Appraisal or evaluation costs

+ Internal and external failure costs

+ Warranty claims and loss of reputation costs

Prevention costs: This is defined as the cost incurred by an organization to prevent the occurrence of errors in the various steps of the development or maintenance process.

– For example, the cost of training employees, the cost of maintenance to ensure a stable manufacturing process, and the cost of making improvements.

Appraisal costs: The cost of verifying or evaluating a product or service during the different steps in the development process. Monitoring system costs (their maintenance and management costs).

Internal failure costs: The cost resulting from anomalies before the product or service has been delivered to the client. Loss of earnings due to non-compliance (cost of making changes, waste, additional human activities, and the use of extra products).

External failure costs: The cost incurred by the company when the client discovers defects. Cost of late deliveries, cost of managing disputes with the client, logistical costs for storing the replacement product or for delivery of the product to the client.

Warranty claims and loss of reputation costs: Cost of warranty execution and damage caused to a corporate image as well as the cost of losing clients.

The first objective of the SQA is to convince management that there are proven benefits to SQA activities.

"identifying an error early in the process can save a lot of time, money and effort."

Figure 2.2 Costs of propagating an error[1] [JON 00].

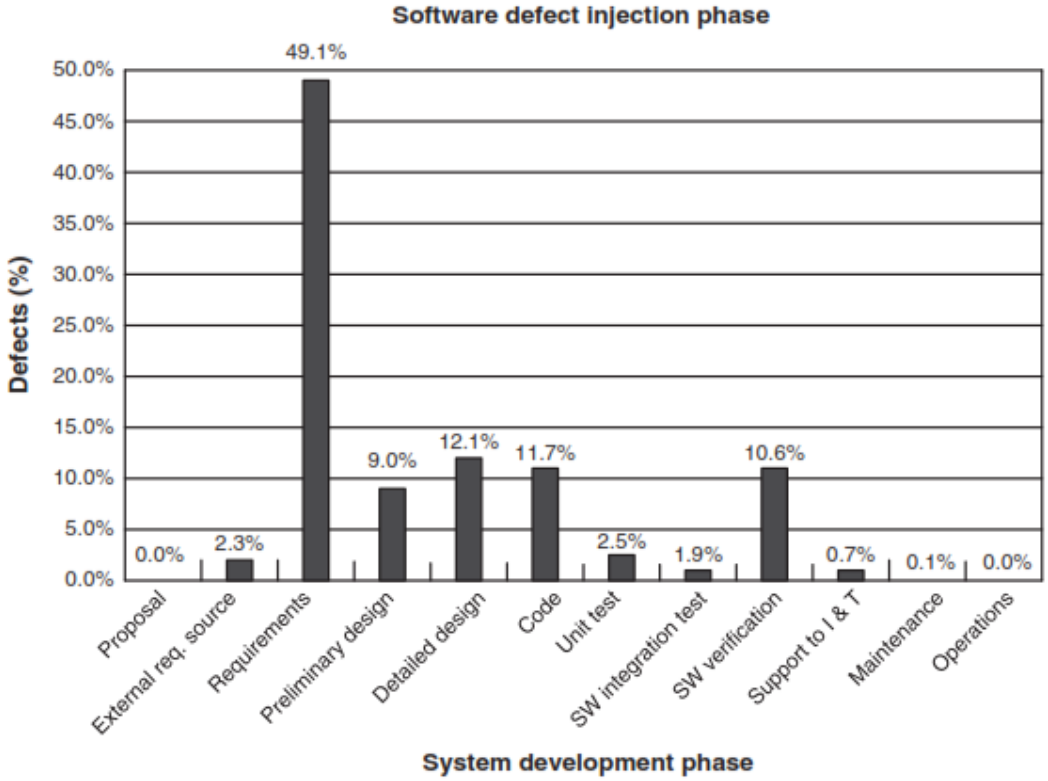


Figure 2.3 Defect injection distribution during the life cycle [SEL 07].

Topic 1.24: QUALITY CULTURE

- Tylor [TYL 10] defined **Human Culture as**
  "that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society."

- It is culture that guides the behaviors, activities, priorities, and decisions of an individual as well as of an organization.
- Wiegers (1996) [WIE 96], in his book "*Creating a Software Engineering Culture,"* illustrates the interaction between the software engineering culture of an organization, its software engineers, and its projects

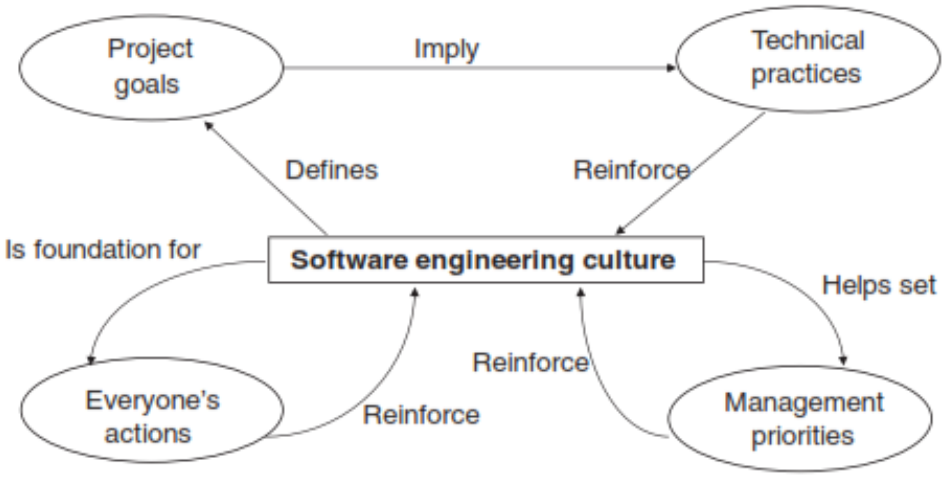


Figure 2.5 Software engineering culture.
Source: Adapted from Wiegers 1996 [WIE 96].

- A healthy culture is made up of the following elements:
  - The personal commitment of each developer to create quality products by systematically applying effective software engineering practices.
  - The commitment to the organization by managers at all levels to provide an environment in which software quality is a fundamental factor of success and allows each developer to carry out this goal.
  - The commitment of all team members to constantly improve the processes they use and to always work on improving the products they create.

Figure 2.6 Start coding … I'll go and see what the client wants!
Source: Reproduced with permission of CartoonStock ltd.

Figure 2.7 Dilbert is threatened and must provide an estimate on the fly. DILBERT © 2007 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

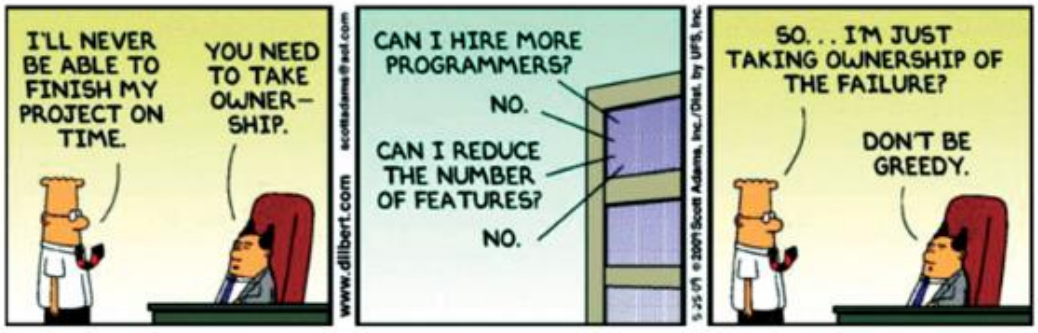


Figure 2.8 Dilbert tries to negotiate a change in his project. DILBERT © 2009 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

Topic 1.25: Fourteen principles to follow to develop a culture that fosters quality

Table 2.3 Cultural Principles in Software Engineering [WIE 96, p. 17]

1. Never let your boss or client cause you to do poor work.
2. People must feel that their work is appreciated.
3. Continuing education is the responsibility of each team member.
4. Participation of the client is the most critical factor of software quality.
5. Your greatest challenge is to share the vision of the final product with the client.
6. Continuous improvement in your software development process is possible and essential.
7. Software development procedures can help establish a common culture of best practices.
8. Quality is the number one priority; long-term productivity is a natural consequence of quality.
9. Ensure that it is a peer, not a client, who finds the defect.
10. A key to software quality is to repeatedly go through all development steps except coding; coding should only be done once.
11. Controlling error reports and change requests is essential to quality and maintenance.
12. If you measure what you do, you can learn to do it better.
13. Do what seems reasonable; do not base yourself on dogma.
14. You cannot change everything at the same time. Identify changes that will reap the most benefits, and start to apply them as of next Monday.

Topic 1.26: THE SOFTWARE ENGINEERING CODE OF ETHICS

- The first draft of the software engineering code of ethics was developed in cooperation with the Institute of Electrical and Electronics Engineers (IEEE) Computer Society and the Association for Computing Machinery (ACM)

Table 2.5 The Eight Principles of the IEEE's Software Engineering Code of Ethics [IEE 99]

Principle	Description
1. The public	Software engineers shall act consistently with the public interest.
2. Client and Employer	Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
3. Product	Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. Judgment	Software engineers shall maintain integrity and independence in their professional judgment.
5. Management	Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. Profession	Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. Colleagues	Software engineers shall be fair to and supportive of their colleagues.
8. Self	Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Topic 1.27: Role of SQA in software development life cycle

- The Software Development Life Cycle is split into six main phases.

1. Planning
2. Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

- In the planning phase of a software project, Quality Assurance is responsible for making sure that the project meets all quality requirements, such as scope, budget, timeline, and compliance with standards.
- QA can also review user requirements and analyze them to determine if they fit within the scope of the project. This helps ensure that expectations are properly set at the beginning of a project, and that resources are allocated appropriately.
- QA is involved in the design phase, they can identify aspects of the design that might cause problems ,while they're still in-progress. This enables the designer or wireframes creator to make changes on the fly.

Topic 1.28: The role QA plays in the implementation stage:

- Code Reviews
- System Integration Testing
- User Acceptance Testing

Topic 1.29: The Role of QA in Testing

QA focuses on various aspects, such as functionality, usability, reliability, performance, and compliance with industry standards. In order to ensure that the requirements of the application are met before it is released to its users.

Topic 1.30: The Role of QA in Deployment

In deployment stage QA ensures that all the elements of specific Custom software development and its properly implemented, tested, verified, and deployed. This allows the product to be released with confidence and without any issues or bugs.

Topic 1.31: The Role of QA in Maintenance

- Verifying software updates to confirm they are working properly
- Testing changes to make sure they are as expected
- Identifying potential problems with updates
- Following up with customers to ensure they are satisfied with changes and updates
- Documenting all issues related to releases, so that future versions can be improved upon accordingly.
- By investing in QA during maintenance, companies can be sure their products remain reliable and bug-free for customers for the long haul!

2. CS 2 & 3 Software Quality Assurance and Testing

Topic 2.1: Standardizing SQA: Quality Models and Management

- Concepts conveyed by software quality models.
- Characteristics and sub-characteristics of software quality
- Software quality requirements of a software product
- Software traceability
- Standards for Quality Management
- Frameworks (ITIL, ISO, CMMI)

Topic 2.2: Requirements

- The needs or requirements of these systems are typically documented either in a request for quote (RFQ) or request for proposal (RFP) document, a statement of work (SOW), a software requirements specification (SRS) document or in a system requirements document (SRD).
    - Using these documents, the software developer must extract the information needed to define specifications for both the functional requirements and performance or non-functional requirements required by the client.

Functional Requirement

A requirement that specifies a function that a system or system component must be able to perform.

ISO 24765 [ISO 17a]

Non-Functional Requirement

A software requirement that describes not what the software will do but how the software will do it. Synonym: design constraint.

ISO 24765 [ISO 17a]

Performance Requirement

The measurable criterion that identifies a quality attribute of a function or how well a functional requirement must be accomplished (IEEE Std 1220™-2005). A performance requirement is always an attribute of a functional requirement.

IEEE 730 [IEE 14]

- Software quality assurance (SQA) must be able to support the practical application of these definitions.
- To achieve this, many concepts proposed by software quality models must be mastered.

Quality Model

A defined set of characteristics, and of relationships between them, which provides a framework for specifying quality requirements and evaluating quality.

ISO 25000 [ISO 14a]

Topic 2.3: Using software quality model client can:

- define software quality characteristics that can be evaluated

- contrast the different perspectives of the quality model that come into play (i.e., internal- process to develop and external-fulfilling requirements perspectives);
- carefully choose a limited number of quality characteristics that will serve as the non-functional requirements for the software (i.e., quality requirements);
- set a measure and its objectives for each of the quality requirements.
- **Evaluation** A systematic examination of the extent to which an entity is capable of fulfilling specified requirements.

Topic 2.4: SOFTWARE QUALITY MODELS

- Unfortunately, in software organizations, software quality models are still rarely used.
- A number of stakeholders have put forth the hypothesis that these models do not clearly identify all concerns for all of the stakeholders involved and are difficult to use.
- Still Formally defining and evaluating the quality of software before it is delivered to the client in a need.

Topic 2.5: Five quality perspectives described by Garvin

- **Transcendental approach to quality:**
  - "Although I can't define quality, I know it when I see it."
  - The main problem with this view is that quality is a personal and individual experience.
  - it only takes time for all users to see it.
- **User-based approach:** A second approach is that quality software performs as expected from the user's perspective (i.e., fitness for purpose).
- **Manufacturing-based approach:** quality is defined as complying with specifications, is illustrated by many documents on the quality of the development process.
- Product-based approach: The product-based quality perspective involves an internal view of the product. The software engineer focuses on the internal properties of the software components, for example, the quality of its architecture. These internal properties correspond to source code characteristics and require advanced testing techniques.
- Value-based approach: focuses on the elimination of all activities that do not add value, for example the drafting of certain documents.

In the software domain, the concept of "value" is synonymous with productivity, increased profitability, and competitiveness.

Topic 2.6: Initial Model Proposed by McCall

It proposes three perspectives for the user and primarily promotes a product-based view of the software product.

– Operation: during its use;

– Revision: during changes made to it over the years;

– Transition: for its conversion to other environments when the time comes to migrate to a new technology.

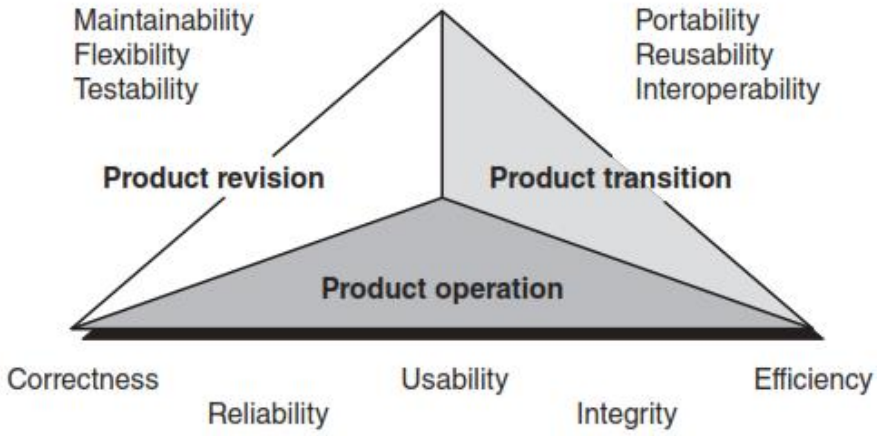


Figure 3.1 The three perspectives and 11 quality factors of McCall et al. (1977) [MCC 77].

- Each perspective is broken down into a number of quality factors.

The model proposed by McCall and his colleagues lists 11 quality factors.

Each quality factor can be broken down into several quality criteria (see Figure 3.2).

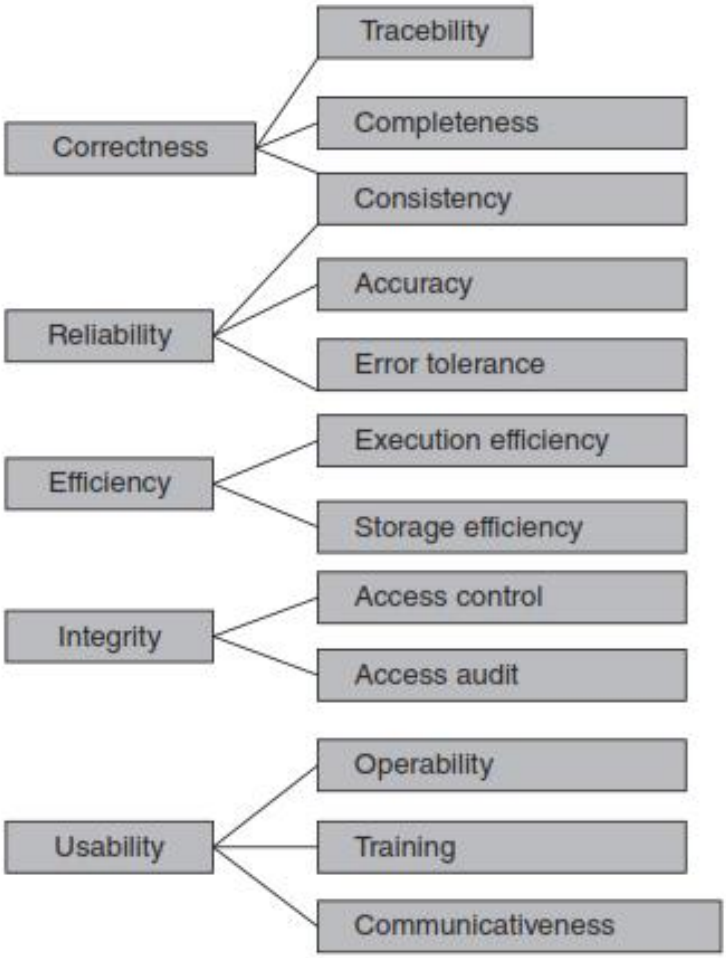


Figure 3.2 Quality factors and criteria from McCall et al. (1977) [MCC 77].

- The right side of Figure 3.2 presents the measurable properties (called "quality criteria"), which can be evaluated (through observation of the software) to assess quality.
- McCall proposes a subjective evaluation scale of 0 (minimum quality) to 10 (maximum quality).
- The McCall quality model was primarily aimed at software product quality (i.e., the internal perspective) and did not easily tie in with the perspective of the user who is not concerned with technical details.
- Example: A car owner who is not concerned with the metals or alloys used to make the engine. He expects the car to be well designed so as to minimize frequent and expensive maintenance costs.

Topic 2.7: The First Standardized Model: IEEE 1061

The IEEE 1061 standard, that is, the *Standard for a Software Quality Metrics Methodology* [IEE 98b], provides a framework for measuring software quality that allows for the establishment and identification of software quality measures based on quality requirements in order to implement, analyze, and validate software processes and products.

This standard claims to adapt to all business models, types of software, and all of the stages of the software life cycle.

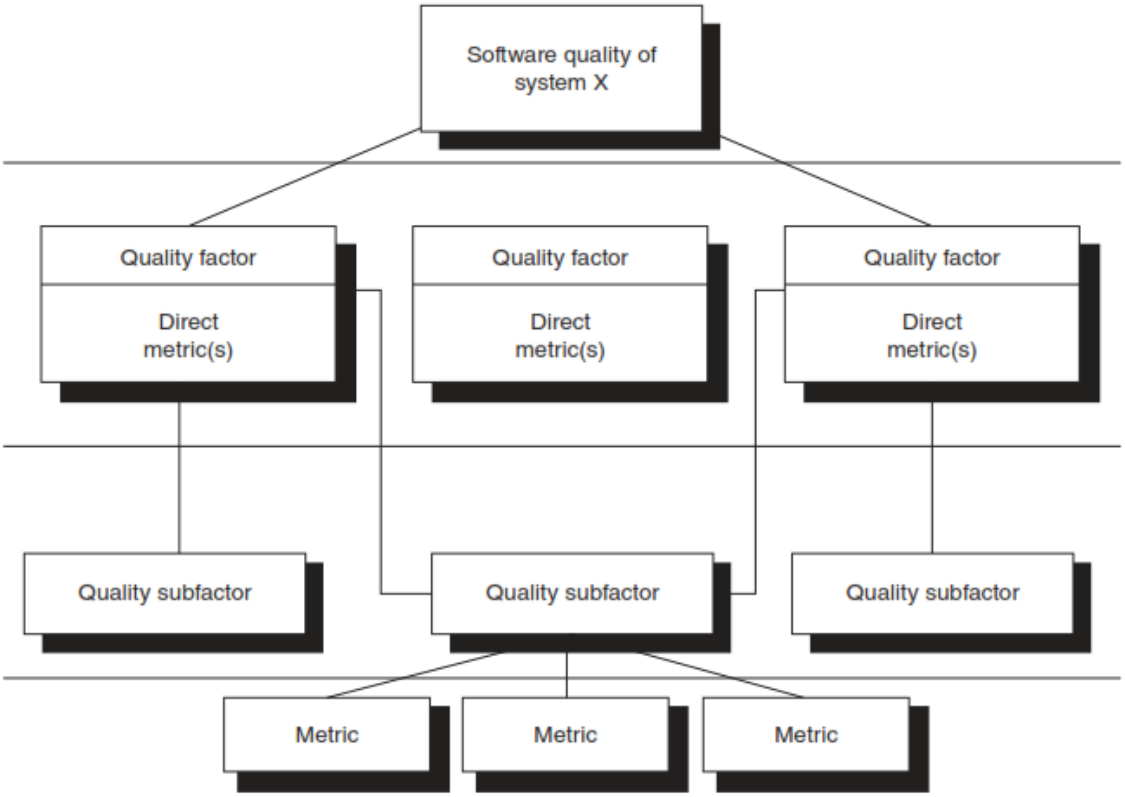


Figure 3.3 Framework for measuring software quality as per the IEEE 1061 [IEE 98b].

At the top tier, we can see that software quality requires prior specification of a certain number of quality attributes, which serve to describe the final quality desired in the software.

The attributes desired by clients and users allow for the definition of the software quality requirements.

Quality factors suggested by this standard are assigned attributes at the next tier.

At the tier below that, and only if necessary, subfactors can then be assigned to each quality factor.

Lastly, measures are associated with each quality factor, allowing for a quantitative evaluation of the quality factor (or subfactor).

As an example, users choose availability as a quality attribute. It is defined in the requirements specifications as being the ability of a software product to maintain a specified level of service when it is used under specific conditions. The team establishes a quality factor, such as mean time between failures (or MTBF).

Users wish to specify that the software should not crash too often, since it needs to perform important activities for the organization. The measurement formula established for Factor A = hours available/(hours available + hours unavailable). It is necessary to identify target values for each directly measured factor. It is also recommended to provide an example of the calculation to clearly define the measure. For example, the work team, when preparing the system specifications, indicates that the MTBF should be 95% to be acceptable (during service hours). If the software must be made available during work hours, that is, 37.5 hours per week, it should therefore not be down for more than 2 hours a week: 37.5/(37.5 + 2) = 0.949%.

Note that if you do not set a target measure (i.e., an objective), there will be no way of determining whether the quality level for the factor was reached when implementing or accepting the software.

Topic 2.8: This model provides defined steps to use quality measures in the following situations:

Software program acquisition: In order to establish a contractual commitment regarding quality objectives for client-users and verify whether they were met by allowing their measurement when adapting and releasing the software.

Software development: In order to clarify and document quality characteristics on which designers and developers must work in order to respect the customer's quality requirements.

Quality assurance/quality control/audit: In order to enable those outside the development team to evaluate the software quality.

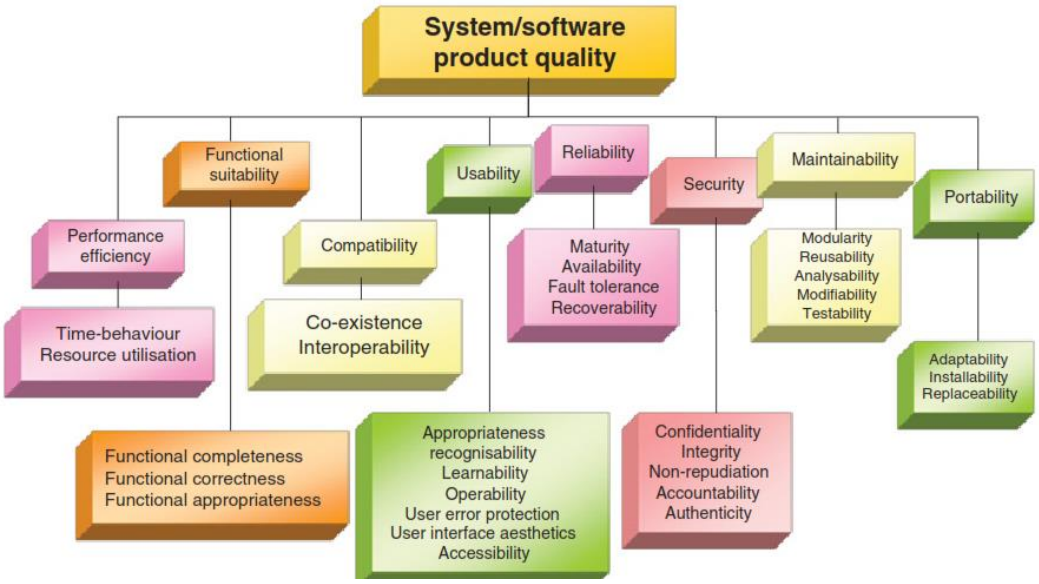
Maintenance: Allow the maintainer to understand the level of quality and service to maintain when making changes or upgrades to the software.

Client/user: Allow users to state quality characteristics and evaluate their presence during acceptance tests (i.e., if the software does not meet the specifications agreed upon by the developer

Topic 2.9: Steps proposed under the IEEE 1061 [IEE 98b] standard:

- Start By Identifying The List Of Non-functional (Quality) Requirements
- Everybody Involved
- List And Make Sure To Resolve Any Conflicting
- Quantify Each Quality Factor.
- Have Measures And Thresholds Approved.
- Perform A Cost–benefit Study To Identify The Costs Of Implementing The Measures For The Project.

- Implement The Measurement Method
- Analyze The Results
- Validate The Measures



Topic 2.10: Current Standardized Model: ISO 25000 Set of Standards

The Treasury Board concluded that there were basically two ways to determine the quality of a software product:

(1) assess the quality of the development process,

(2) assess the quality of the final product.

The ISO 25000 [ISO 14a] standard allows for the evaluation of the quality of the final software product.

The ISO 25000's series of standards recommends the following four steps [ISO 14a]:

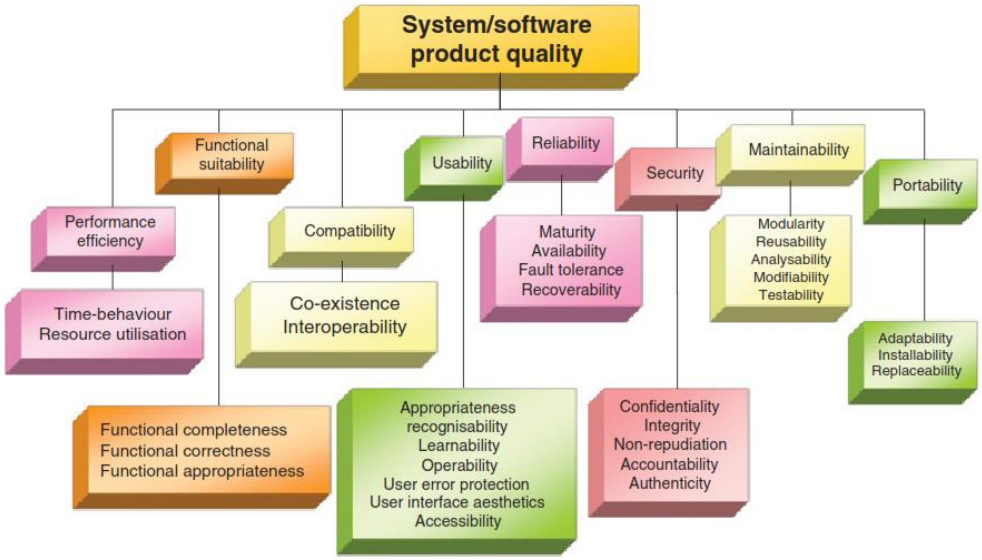
– Set quality requirements;

– Establish a quality model;

– Define quality measures;

– Conduct evaluations.

The ISO 25010 standard identifies eight quality attributes for software



To illustrate how this standard is used, we will describe the characteristic of maintainability, which has five sub-characteristics: modularity, reusability, analyzability, modifiability, and testability.

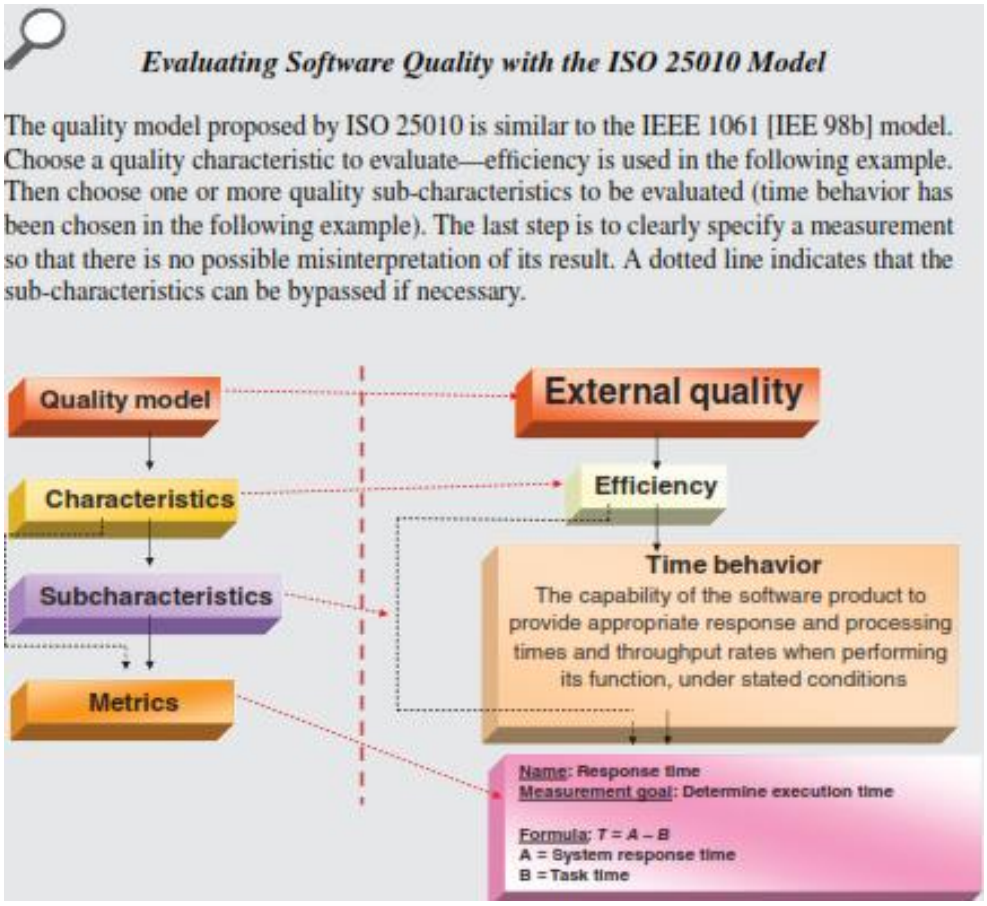
Maintainability is defined as being the level of efficiency and efficacy with which software can be modified. Changes may include software corrections, improvements, or adaptation to changes in the environment, requirements, or functional specifications.

The internal and external points of view, of the maintainability of the software.

External point of view, maintainability attempts to measure the effort required to troubleshoot, analyze, and make changes to specific software.

Internal point of view, maintainability usually involves measuring the attributes of the software that influence this change effort.

Maintainability	Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers
• Modularity	Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components
• Reusability	Degree to which an asset can be used in more than one system, or in building other assets
• Analyzability	Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified
• Modifiability	Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality
• Testability	Degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component and tests can be performed to determine whether those criteria have been met



Topic 2.11: DEFINITION OF SOFTWARE QUALITY REQUIREMENTS

Process of defining quality requirements for software (i.e., a process that supports the use of a software quality model).

In the classical engineering approach, requirements are considered to be prerequisites to the design and development stages of a product.

The requirements development phase may have been preceded by a feasibility study, or a design analysis phase for the project.

Once the stakeholders have been identified, activities for software specifications can be broken down into:

– gather: collect all wishes, expectations, and needs of the stakeholders;

– prioritize: debate the relative importance of requirements based on, for example, two priorities (essential, desirable);

– analyze: check for consistency and completeness of requirements;

– describe: write the requirements in a way that can be easily understood by users and developers;

– specify: transform the business requirements into software specifications (data sources, values and timing, business rules).

Requirements are generally grouped into three categories:

1) Functional Requirements: These describe the characteristics of a system or processes that the system must execute. This category includes business requirements and functional requirements for the user.

2) Non-Functional (Quality) Requirements: These describe the properties that the system must have, for example, requirements translated into quality characteristics and sub-characteristics such as security, confidentiality, integrity, availability, performance, and accessibility.

3) Constraints: Limitations in development such as infrastructure on which the system must run or the programming language that must be used to implement the system.

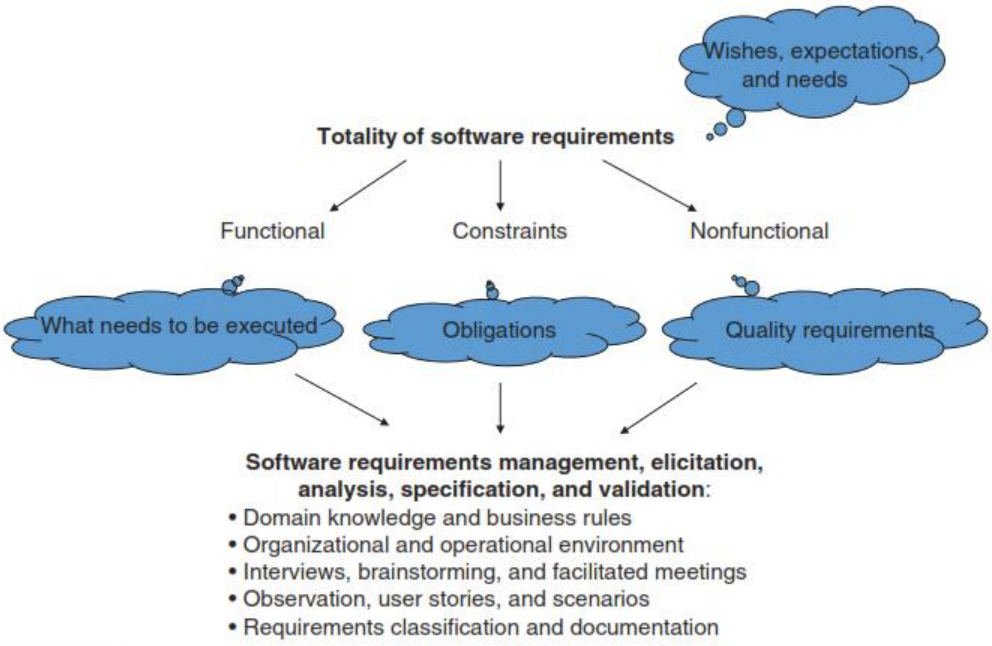


Figure 3.5 Context of software requirements elicitation.

Topic 2.12: Characteristics to measure quality of a requirement

- Necessary: They must be based on necessary elements
- Unambiguous: clear enough to be interpreted in only one way.
- Concise: They must be stated in a language that is precise, brief, and easy to read.
- Coherent: They must not contradict the requirements.
- Complete: They must all be stated fully
- Accessible: They must be realistic regarding their implementation (time ,budget ,resource)
- Verifiable: inspection, analysis, demonstration, or tests.

Topic 2.13: Specifying Quality Requirements: The Process

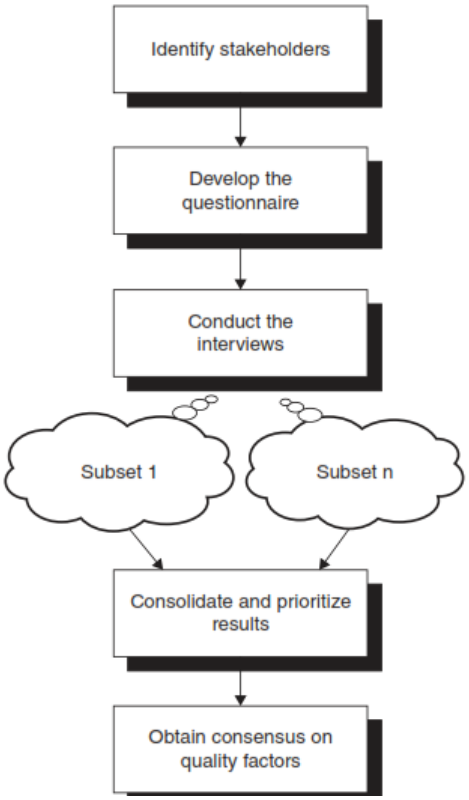


Figure 3.6 Steps suggested for defining

- Stakeholders are any person or organization that has a legitimate interest in the quality of the software.

These needs and hopes may change during the system life cycle and must be checked when there is any change.

- Developing the questionnaire that presents the external quality characteristics in terms that are easy to understand for managers.

Table 3.3 Example of Quality Criteria Documentation

Quality characteristics	Importance
Reliability	Indispensable
User-friendliness	Desirable
Operational safety	Non-applicable

Next, for each characteristic, the quality measure must be described in detail and include the following information

– quality characteristic;
– quality sub-characteristic;
– measure (i.e., formula);
– objectives (i.e., target);
– example.

The last step in defining quality requirements involves having these requirements authorized through consensus.

Evaluation of the Functional Capacity of Software

Users often choose this characteristic. It is defined in the specifications as the ability of a software product to carry out all specified requirements. The *ability* sub-characteristic is chosen by the team and described as the percentage of requirements described in the specification document that must be delivered (*%E*). The measure established is

$$\%E = (\text{Number of functionalities requested}/(\text{Number of functionalities delivered})) \times 100.$$

It is necessary to identify target values as an objective for each measure. It is also recommended to provide an example of the calculations (i.e., measurement) to clearly illustrate the measure. For example, during the writing of the specification, the project team indicates that the *%E* should be 100% of the requirements described in the specifications document and that they are functional and delivered, without defects, before the final acceptance of the software for production.

Alternatively, with a lack of measurable objectives, the general policy is to accept the most stable version of the code having the necessary functionality.

ISO 25010 [ISO 11i]

Topic 2.14: REQUIREMENT TRACEABILITY DURING THE SOFTWARE LIFE CYCLE

Throughout the life cycle, client needs are documented and developed in different documents, such as specifications, architecture, code, and user manuals.

Throughout the life cycle of a system, many changes regarding client needs should be expected.

Every time a need changes, it must be ensured that all documents are updated.

Traceability is a technique that helps us follow the development of needs as well as their changes.

Topic 2.15: Software Engineering Standards

Other engineering domains such as mechanical, chemical, electrical, or physics engineering are based on the laws of nature as discovered by scientists.

Hooke's Law

$$\sigma = E \cdot \varepsilon$$

Gravitational Law

$$\vec{F}_{A\to B} = -G\frac{M_A M_B}{AB^2}\vec{u}_{AB}$$

Newton's Law

$$x(t) = \frac{1}{2}a \cdot t^2 + v_0 \cdot t + x_0$$

Ohm's Law

$$V = RI$$

Boyle–Mariotte's Law

$$p_1 x V_1 = p_2 x V_2$$

Curie's Law

$$E = -\vec{\mu} \cdot \vec{B}$$

Coulomb's law

$$F_{12} = \frac{q_1 q_2}{4\pi\epsilon_0}\frac{r_2 - r_1}{|r_2 - r_1|^3}$$

Refraction Law

$$\eta_1 \cdot \sin(\theta_1) = \eta_2 \cdot \sin(\theta_2)$$

Figure 4.1 A few laws of nature used by some engineering disciplines.

Unfortunately, software engineering, unlike other engineering disciplines, is not based on the laws of nature.

Software engineering, like other disciplines, is based on the use of well-defined practices for ensuring the quality of its products.

In software engineering, there are several standards, which are actually guides for management practices.

A rigorous process is the framework for the way standards are developed and approved including, among others, international ISO standards and standards from professional organizations such as IEEE.

Standard

A set of mandatory requirements established by consensus and maintained by a recognized body to prescribe a disciplined and uniform approach, or to specify a product, with respect to mandatory conventions and practices.

The four principles for the development of ISO standards are:

– ISO standards meet a market need.

– ISO standards are based on worldwide expertise.

– ISO standards are the result of a multi-stakeholder process.

– ISO standards are based on consensus.

The ISO standards are developed by consensus

– That all parties were able to express their views;

– The best effort has been made to take into account all opinions and solve all problems (i.e., all the submissions in a vote of the draft of a standard).

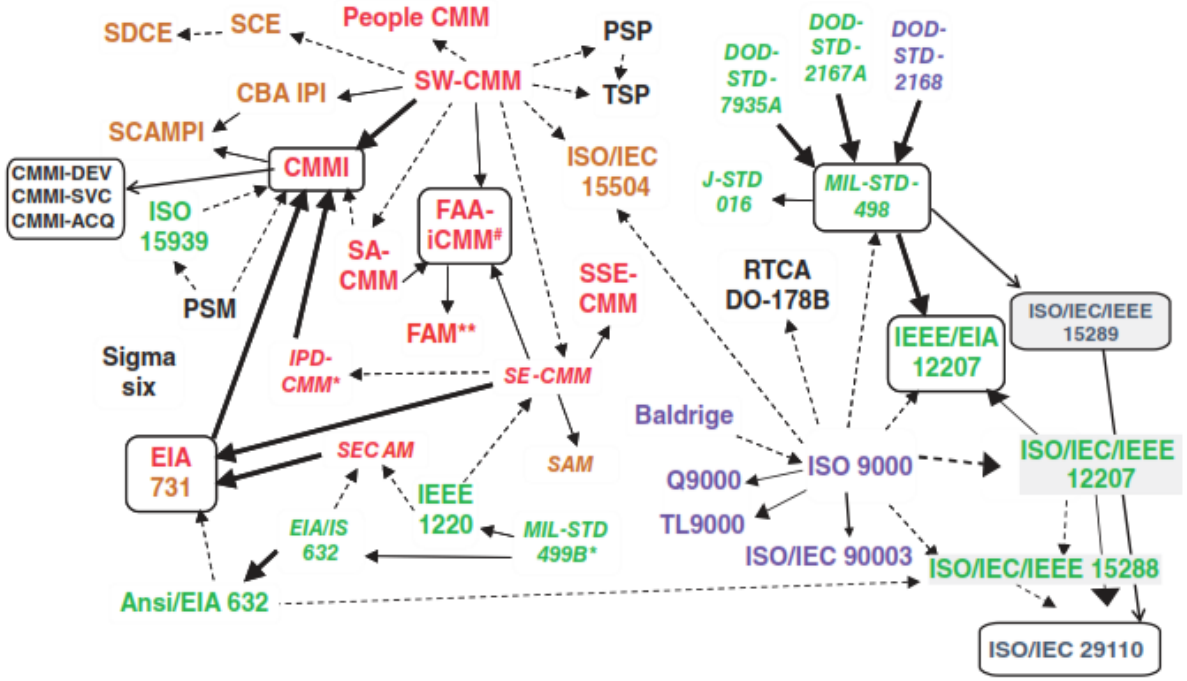


Figure 4.2 The development of standards and models.

- American Department of Defence (DoD) created the "DoD-STD1679A" military standard
- IEEE, the International Organization for Standardization (ISO)

European Space Agency (ESA) developed standards

- "*Capability Maturity Model"* (CMM®): developed at the request of the American DoD, by the Software Engineering Institute (SEI) in order to provide a road map of engineering practices to improve the performance of the development, maintenance and service provisioning processes.

Figure 4.3 illustrates the evolution of standards that are maintained and published under the responsibility of the appointed subcommittee for standardized processes, tools, and supporting technologies for software engineering and systems:

Topic 2.16: The Continuous Evolution of Standards

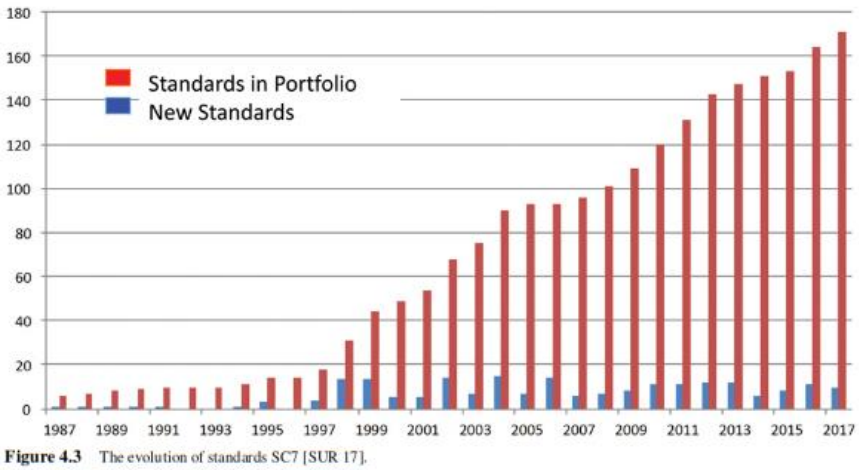


Figure 4.3 The evolution of standards SC7 [SUR 17].

Topic 2.17: MAIN STANDARDS FOR QUALITY MANAGEMENT

Standards related to the management of software quality:

ISO 9000 [ISO 15b] and ISO 9001 [ISO 15].

The application guide for software, The ISO/IEC 90003 standard.

Standards in the ISO 9000 family include:

- ISO 9001:2015 - sets out the requirements of a quality management system
- ISO 9000:2015 - covers the basic concepts and language
- ISO 9004:2009 - focuses on how to make a quality management system more efficient and effective
- ISO 19011:2011 - sets out guidance on internal and external audits of quality management systems.

- The ISO 9001 standard provides the basic concepts, principles and vocabulary of quality management systems (QMS) and is the basis for other standards for QMSs [ISO 15].
- The "Quality Management Principles" (QMP) are a set of values, rules, standards, and fundamental convictions regarded as fair and that could be the basis for quality management.

Topic 2.18: The seven QMP of the ISO 9001

– Principle 1: Customer focus

– Principle 2: Leadership

– Principle 3: Involvement of people

– Principle 4: Process approach

– Principle 5: System approach to management

– Principle 6: Factual approach to decision making

– Principle 7: Mutually beneficial supplier relationships

Topic 2.19: ISO 9001 uses the process approach, the Plan-Do-Check-Act (PDCA) approach, and a risk-based thinking approach [ISO 15].

– The process approach allows an organization to plan its processes and their interactions.

– The PDCA cycle allows an organization to ensure that its processes are adequately resourced and appropriately managed and that opportunities for improvement are identified and implemented.

– The risk-based thinking approach allows an organization to determine the factors that may cause deviation from its processes and its QMS in relation to expected results, to implement preventive measures in order to limit negative effects and exploit opportunities when they arise.

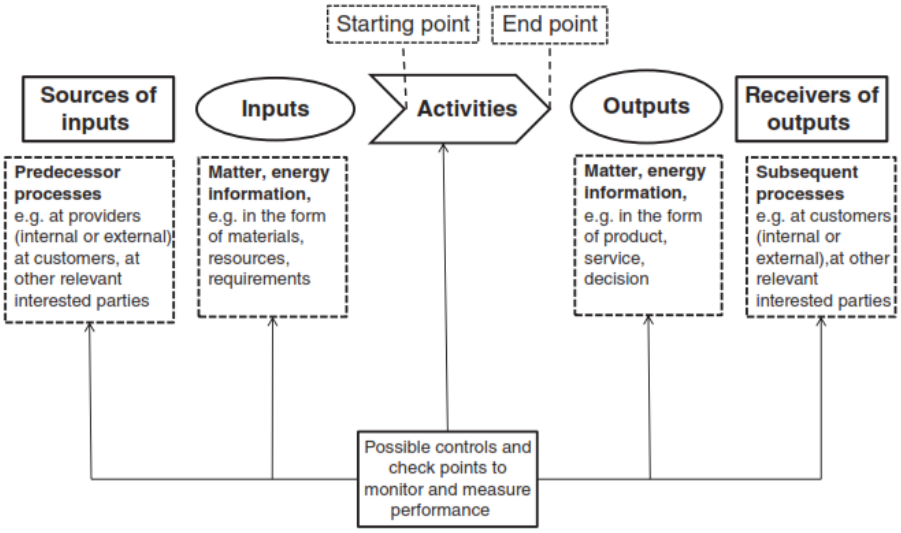


Figure 4.4 Elements of a process [ISO 15].

ISO 9001 describes the elements of the PDCA cycle as follows :

– Plan: establish the objectives of the system, processes and resources to deliver results in accordance with customer requirements and policies of the organization, identify and address risks and opportunities;

– Do: implement what has been planned;

– Check: monitor and measure (if applicable) processes and the products and services obtained against policies, objectives, requirements and planned activities, and report the results;

– Act: take actions to improve performance, as needed.

Topic 2.20: ISO/IEC 90003 Standard

International Electrotechnical Commission.

- provides guidelines for the application of the ISO 9001 standard to computer software.
- It provides organizations with instructions for acquiring, supplying, developing, using and maintaining software.
- Explains what a software audit is for the organization wishing to set up a QMS as well as for the QMS auditor.

Topic 2.21: ISO/IEC/IEEE 12207 STANDARD

Establishes a common framework for software life cycle processes.

It applies to the acquisition of systems and software products and services, supply, development, operation, maintenance, and disposal of software products and the development of the software part of a system whether performed internally or externally to an organization

ISO 12207 [ISO 17] defines four sets of processes as shown in Figure 4.5:

– Two agreement processes between a customer and a supplier;

– Six organizational project-enabling processes;

– Eight processes for technology management;

– **Fourteen technical processes.**

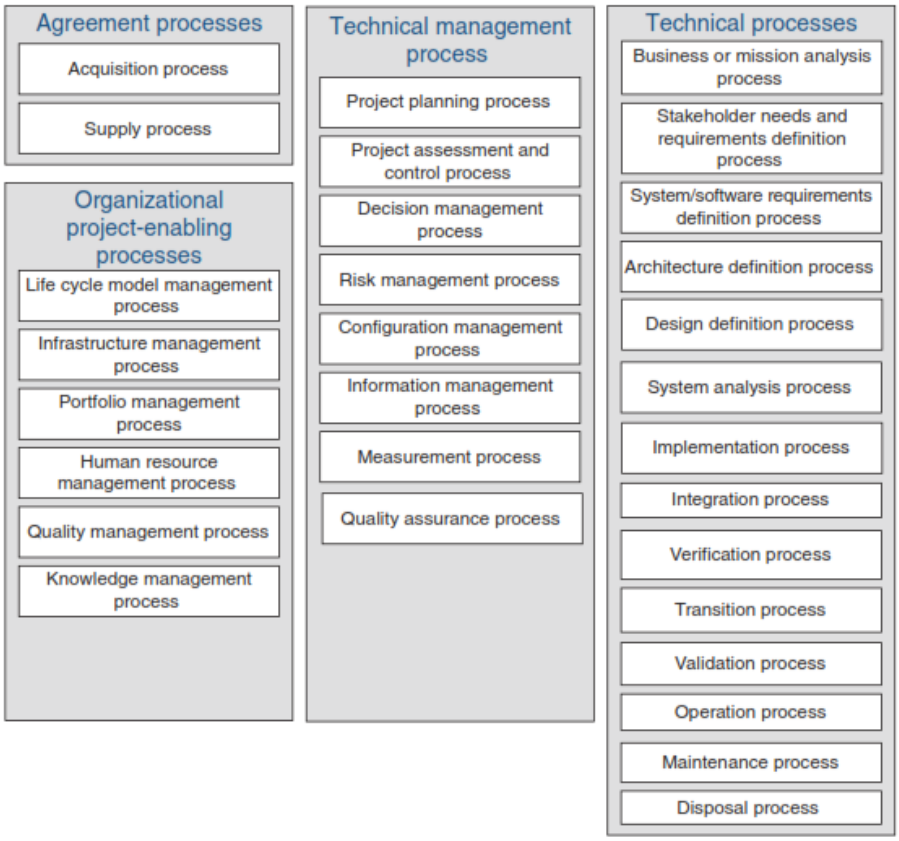


Figure 4.5 The four life cycle process groups of ISO 12207 [ISO 17].

Topic 2.22: The ISO 12207 standard can be used in one or more of the following modes

By an organization: to help establish an environment of desired processes. These processes can be supported by an infrastructure of methods, procedures, techniques, tools, and trained personnel.

By a project: to help select, structure and employ the elements of an established set of life cycle processes to provide products and services.

By an acquirer and a supplier: to help develop an agreement concerning processes and activities.

By organizations and assessors: to serve as a process reference model for use in the performance of process assessments that may be used to support organizational process improvement.

Topic 2.23: IEEE 730 STANDARD FOR SQA PROCESSES

QA according to IEEE is a set of proactive measures to ensure the quality of the software product.

The IEEE 730 provides guidance for the SQA activities of products or of services.

The SQA process of the IEEE 730 is grouped into three activities: the implementation of the SQA process, product assurance, and process assurance.

Activities consist of a set of tasks.

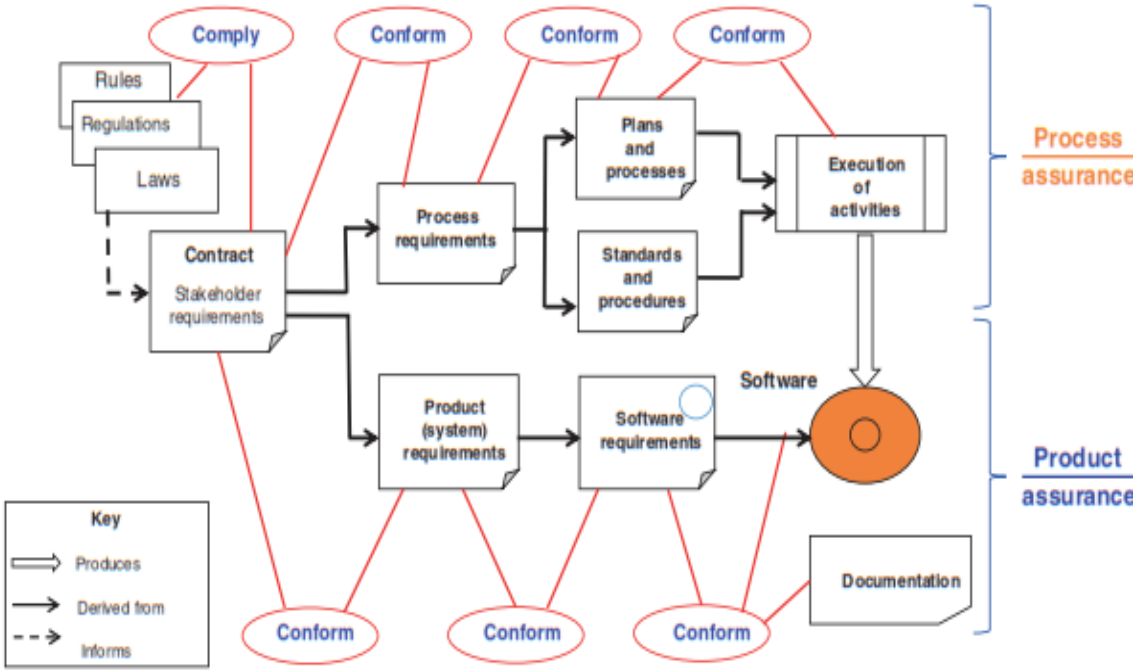


Figure 4.6 The links between requirements and the artifacts of a project [IEE 14].

IEEE 730 [IEE 14] describes what must be done by a project;

- it assumes that the organization has already implemented SQA processes before the start of a project.
- The standard includes a clause that describes what is meant by compliance.

Topic 2.24: Product Assurance Activities

– evaluate plans for compliance to contracts, standards, and regulations;

– evaluate product for compliance to established requirements;

– evaluate product for acceptability;

– evaluate the compliance of product support;

– measure products.

Topic 2.25: Process Assurance Activities

– evaluate compliance of the processes and plans;

– evaluate environments for compliance;

– evaluate subcontractor processes for compliance;

– measure processes;

– assess the skill and knowledge of personnel.

Topic 2.26: Capability Maturity Models (CMM®).

- Tool used to improve and refine software development processes.

- Framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to enhance further the maturity of the process used to develop those software products.

Topic 2.27: The Capability Maturity Model Integration (CMMI)

- An advanced framework designed to improve and integrate processes across various disciplines such as software engineering, systems engineering, and people management.
- Helps organizations fulfill customer needs, create value for investors, and improve product quality and market growth.
- The CMMI model was developed as two versions:
- Initial staged version and continuous version, which is the first CMM model for systems engineering
- **CMMI-DEV** The objective of this model is to encourage organizations to check and continuously improve their development project process and evaluate their level of maturity on a five-level scale as proposed by the staged CMMI model.
- The **CMMI for Development (CMMI-DEV**) covers a broader area than its predecessor by adding other practices, such as systems engineering, and the development of integrated processes and products.
- The objective of this model is to encourage organizations to check and continuously improve their development project process and evaluate their level of maturity on a five-level scale
- Two other CMMI models were developed based on architecture, CMMI for Services (CMMI-SVC) [SEI 10b] and the CMMI for Acquisition (CMMIACQ) [SEI 10c].
- The CMMI-SVC model provides guidelines for organizations that provide services either internally or externally.
- The CMMI-ACQ model provides guidelines for organizations that purchase products or services.
- All three CMMI models use 16 common process areas.
- For each level of maturity, a set of process areas are defined.
- Each area encompasses a set of requirements that must be met.
- These requirements define which elements must be produced rather than *how they are produced*

Thereby allowing the organization implementing the process to choose its own life cycle model, its design methodologies, its development tools, its programming languages, and its documentation standard.

This approach enables a wide range of companies to implement this model while having processes that are compatible with other standards.

Topic 2.28: Maturity levels and process areas for each maturity level in the CMMI-DEV model.

Maturity Level 1: Initial

Processes are usually ad hoc and chaotic.

Maturity level 1 organizations are characterized by a tendency to overcommit, abandon their processes in a time of crisis, and be unable to repeat their successes.

Maturity Level 2: Managed

When these practices are in place, projects are performed and managed according to their documented plans.

Process areas:

- Requirements management
- Project planning
- Project monitoring and control
- Supplier agreement management
- Measurement and analysis
- Process and product quality assurance
- Configuration management

Maturity Level 3: Defined

Processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

Process areas:

- Requirements development
- Technical solution
- Product integration
- Verification
- Validation
- Organizational process focus
- Organizational process definition
- Organizational training integrated project management
- Risk management
- Decision analysis and resolution

Maturity Level 4: Quantitatively managed

The organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing projects.

Process areas:

- Organizational process performance
- Quantitative project management

Maturity Level 5: Optimizing

An organization continually improves its processes based on a quantitative understanding of its business objectives and performance needs.

Process areas

- Organizational performance management
- Causal analysis and resolution

Topic 2.29: CMMI model structure.

Each process area has generic and specific goals, practices, and sub-practices.

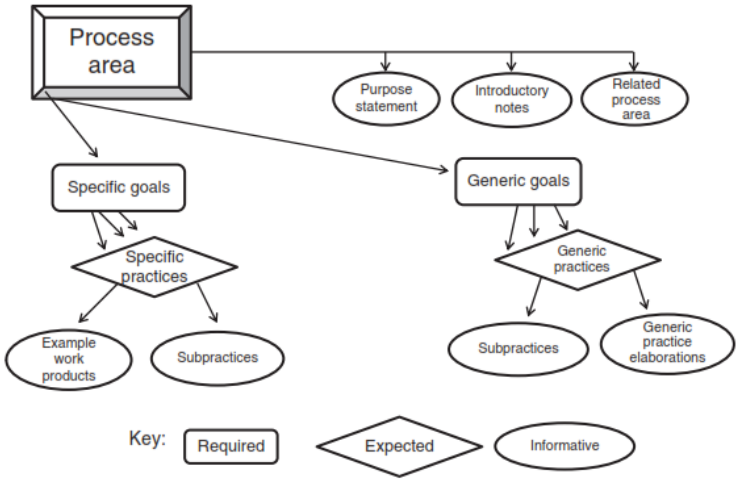


Figure 4.8 Structure of the staged representation of the CMMI [SEI 10a].

Level	Focus	Key process area	Quality productivity
5 Optimizing	*Continuous process improvement*	Organizational performance management causal analysis and resolution	
4 **Quantitatively managed**	*Quantitative management*	Organizational process performance quantitative project management	
3 **Defined**	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Risk management Decision analysis and resolution	
2 Managed	*Basic project management*	Requirement management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management	Risk rework
1 Initial			

Figure 4.9 The staged representation of the CMMI® for Development model.

Topic 2.30: ITIL Framework

The ITIL framework was created in Great Britain based on good management practices for computer services.

It consists of a set of five books providing advice and recommendations in order to offer quality service to IT service users.

IT services are typically responsible for ensuring that the infrastructures are effective and running (backup copies, recovery, computer administration, telecommunications, and production data)

– strategy;

– design;

– transition;

– operation;

– continuous improvement.

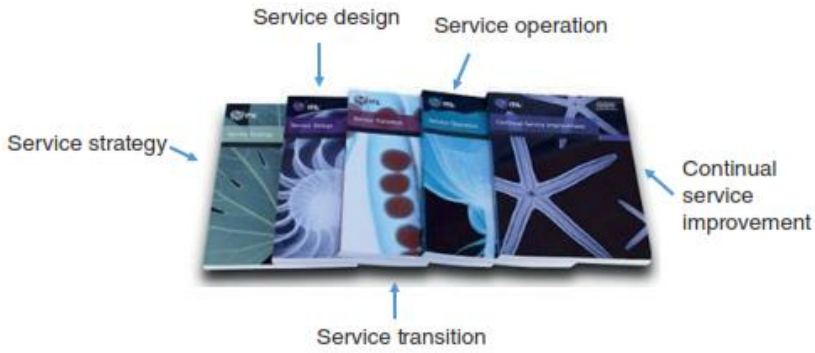


Figure 4.11 The main ITIL guides.

The ITIL framework offers guidance and best practices for managing the five stages of the IT service lifecycle: service strategy, service design, service transition, service operation and continual service improvement.

Support processes described in the ITIL are focused on daily operations. Their main goals are to resolve the problems when they arise or to prevent them from happening when there is a change in the computer environment or in the way the organization does things.

Topic 2.31: Support center function

– Incident management

– Problem management

– Configuration management

– Change management

– Commissioning management

Topic 2.32: Five processes for service operation:

– Service level management

– Financial management of IT services

– Capacity management

– IT service continuity management

– Availability management

Given the major recognition of ITIL worldwide, an international standard based on ITIL came into being: ISO/IEC 20000-1.

The principles of ITIL were successfully conveyed to many companies of all sizes and from all sectors of activity

Topic 2.33: The main subjects handled under ITIL

– user support, which includes the management of incidents and is an extension of the concept of a Helpdesk;

– provision of services which involves managing processes that are dedicated to the daily operations of IT (cost control, management of service levels);

– management of the production environment infrastructure which involves implementing the means for network management and production tools (scheduling, backup, and monitoring);

– application management which consists of managing the support of an operational program;

– security management (confidentiality, data integrity, data availability, etc.) of the SI (security process)

3. CS4 Software Quality Assurance and Testing

Topic 3.1: *The need for a comprehensive definition of requirements*

Cover all attributes of software and aspects of the use of software, including usability aspects, reusability aspects, maintainability aspects, and so forth in order to assure the full satisfaction of the users.

The great variety of issues related to the various attributes of software and its use and maintenance, as defined in software requirements documents, can be classified into content groups called *quality factors.*

Topic 3.2: Classifications of software requirements into software quality factors

The classic model of software quality factors, suggested by McCall, consists of 11 factors.

Subsequent models, consisting of 12 to 15 factors

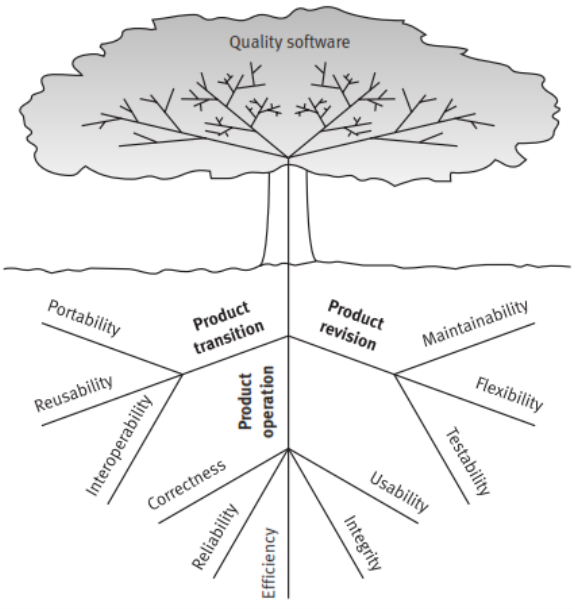
The McCall factor model, despite the quarter of a century of its "maturation", continues to provide a practical, up-to-date method for classifying software requirements

Topic 3.3: McCall's factor model

Classifies all software requirements into 11 software

quality factors.

- Product operation factors: Correctness, Reliability, Efficiency, Integrity, Usability.
- Product revision factors: Maintainability, Flexibility, Testability.
- Product transition factors: Portability, Reusability, Interoperability.



1: McCall's factor model tree

Topic 3.4: Product operation software quality factors

Correctness : Correctness requirements are defined in a list of the software system's required outputs.

Query display of a customer's balance in the sales accounting information system

Air supply as a function of temperature specified by the firmware of an industrial control unit.

Topic 3.5: Output specifications are usually multidimensional

- The output mission (e.g., sales invoice printout, and red alarms when temperature rises above 250°F).
- The required accuracy of those outputs that can be adversely affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be adversely affected by incomplete data.
- The up-to-dateness of the information (defined as the time between the event and its consideration by the software system).
- The availability of the information (the reaction time, defined as the time needed to obtain the requested information or as the requested reaction time of the firmware installed in a computerized apparatus).
- The standards for coding and documenting the software system.

Example

The correctness requirements of a club membership information system consisted of the following:

- The output mission: A defined list of 11 types of reports, four types of standard letters to members and eight types of queries, which were to be displayed on the monitor on request.
- The required accuracy of the outputs: The probability for a non-accurate output, containing one or more mistakes, will not exceed 1%.
- The completeness of the output information: The probability of missing data about a member, his attendance at club events, and his payments will not exceed 1%.
- The up-to-dateness of the information: Not more than two working days for information about participation in events and not more than one working day for information about entry of member payments and personal data.
- The availability of information: Reaction time for queries will be less than two seconds on average; the reaction time for reports will be less than four hours.
- The required standards and guidelines: The software and its documentation are required to comply with the client's guidelines.

Reliability

Reliability requirements deal with failures to provide service.

They determine the maximum allowed software system failure rate, and can refer to the entire system or to one or more of its separate functions.

Example

(1) The failure frequency of a heart-monitoring unit that will operate in a hospital's intensive care ward is required to be less than one in 20 years. Its heart attack detection function is required to have a failure rate of less than one per million cases.

(2) One requirement of the new software system to be installed in the main branch of Independence Bank, which operates 120 branches, is that it will not fail, on average, more than 10 minutes per month during the bank's office hours. In addition, the probability that the off-time (the time needed for repair and recovery of all the bank's services) be more than 30 minutes is required to be less than 0.5%.

Efficiency

Efficiency requirements deal with the hardware resources needed to perform all the functions of the software system in conformance to all other requirements.

computer's processing capabilities, data storage capability, data communication capability of the communication lines

Another type of efficiency requirement deals with the time between recharging of the system's portable units, such as, information systems units located in portable computers, or meteorological units placed outdoors.

Examples

(1) A chain of stores is considering two alternative bids for a software system. Both bids consist of placing the same computers in the chain's headquarters and its branches. The bids differ solely in the storage volume: 20 GB per branch computer and 100 GB in the head office computer (Bid A); 10 GB per branch computer and 30 GB in the head office computer (Bid B). There is also a difference in the number of communication lines required: Bid A consists of three communication lines of 28.8 KBPS between each branch and the head office, whereas Bid B is based on two communication lines of the same capacity between each branch and the head office. In this case, it is clear that Bid B is more efficient than Bid A because fewer hardware resources are required.

Integrity

Integrity requirements deal with the software system security, that is, requirements to prevent access to unauthorized persons, to distinguish between the majority of personnel allowed to see the information ("read permit") and a limited group who will be allowed to add and change data ("write permit"), and so forth.

Example

The Engineering Department of a local municipality operates a GIS (Geographic Information System). The Department is planning to allow citizens access to its GIS files through the Internet. The software requirements include the possibility of viewing and copying but not inserting changes in the maps of their assets as well as any other asset in the municipality's area ("read only" permit). Access will be denied to plans in progress and to those maps defined by the Department's head as limited access documents.

Usability

Usability requirements deal with the scope of staff resources needed to train a new employee and to operate the software system.

Example

The software usability requirements document for the new help desk system initiated by a home appliance service company lists the following specifications:

(a) A staff member should be able to handle at least 60 service calls a day.

(b) Training a new employee will take no more than two days (16 training hours), immediately at the end of which the trainee will be able to handle 45 service calls a day.

Topic 3.6: Product revision software quality factors

Maintainability

Maintainability requirements determine the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.

This factor's requirements refer to the modular structure of software, the internal program documentation, and the programmer's manual, among other items.

Example

Typical maintainability requirements:

(a) The size of a software module will not exceed 30 statements.

(b) The programming will adhere to the company coding standards and guidelines.

Flexibility

The capabilities and efforts required to support adaptive maintenance activities are covered by the flexibility requirements.

These include the resources (i.e. in man-days) required to adapt a software package to a variety of customers of the same trade, of various extents of activities, of different ranges of products and so on.

Example

TSS (teacher support software) deals with the documentation of pupil achievements, the calculation of final grades, the printing of term grade documents, and the automatic printing of warning letters to parents of failing pupils. The software specifications included the following flexibility requirements:

(a) The software should be suitable for teachers of all subjects and all school levels (elementary, junior and high schools).

(b) Non-professionals should be able to create new types of reports according to the schoolteacher's requirements and/or the city's education department demands.

Testability

Testability requirements deal with the testing of an information system as well as with its operation.

Testability requirements for the ease of testing are related to special features in the programs that help the tester, for instance by providing predefined intermediate results and log files.

Example

An industrial computerized control unit is programmed to calculate various measures of production status, report the performance level of the machinery, and operate a warning signal in predefined situations.

One testability requirement demanded was to develop a set of standard test data with known system expected correct reactions in each stage. This standard test data is to be run every morning, before production begins, to check whether the computerized unit reacts properly.

Topic 3.7: Product transition software quality factors

Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth.

Example

A software package designed and programmed to operate in a Windows 2000 environment is required to allow low-cost transfer to Linux and Windows NT environments.

Reusability

Reusability requirements deal with the use of software modules originally designed for one project in a new software project currently being developed.

They may also enable future projects to make use of a given module or a group of modules of the currently developed software.

The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

These benefits of higher quality are based on the assumption that most of the software faults have already been detected by the quality assurance activities performed on the original software, by users of the original software, and during its earlier reuses.

Example

A software development unit has been required to develop a software system for the operation and control of a hotel swimming pool that serves hotel guests and members of a pool club. Although the management did not define any reusability requirements, the unit's team leader, after analyzing the information processing requirements of the hotel's spa, decided to add the reusability requirement that some of the software modules for the pool should be designed and programmed in a way that will allow its reuse in the spa's future software system, which is planned to be developed next year.

These modules will allow:

- Entrance validity checks of membership cards and visit recording.
- Restaurant billing.
- Processing of membership renewal letters.

Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware.

Interoperability requirements can specify the name(s) of the software or firmware for which interface is required.

Example

The firmware of a medical laboratory's equipment is required to process its results (output) according to a standard data structure that can then serve as input for a number of standard laboratory information systems.

Topic 3.8: Alternative models of software quality factors

- The Evans and Marciniak factor model (Evans and Marciniak, 1987).
- The Deutsch and Willis factor model (Deutsch and Willis, 1988).

Topic 3.9: Comparison of the alternative models

Both alternative models exclude only one of McCall's 11 factors, namely the testability factor.

- The Evans and Marciniak factor model consists of 12 factors that are classified into three categories.
- The Deutsch and Willis factor model consists of 15 factors that are classified into four categories.

Taken together, five new factors were suggested by the two alternative factor models.

- Verifiability (by both models)

- Expandability (by both models)
- Safety (by Deutsch and Willis)
- Manageability (by Deutsch and Willis)
- Survivability (by Deutsch and Willis).

Table 3.1: Comparison of McCall's factor model and alternative models

No.	Software quality factor	McCall's classic model	Alternative factor models	
			Evans and Marciniak	Deutsch and Willis
1	Correctness	+	+	+
2	Reliability	+	+	+
3	Efficiency	+	+	+
4	Integrity	+	+	+
5	Usability	+	+	+
6	Maintainability	+	+	+
7	Flexibility	+	+	+
8	Testability	+		
9	Portability	+	+	+
10	Reusability	+	+	+
11	Interoperability	+	+	+
12	Verifiability		+	+
13	Expandability		+	+
14	Safety			+
15	Manageability			+
16	Survivability			+

4. CS5 & CS6 Software Quality Assurance and Testing

Topic 4.1: Deep driving SQA: Software Testing Techniques

Software testing (or "testing") was the first software quality assurance tool applied to control the software product's quality before its shipment or installation at the customer's premises.

SQA professionals were encouraged to extend testing to the partial in-process products of coding, which led to software module (unit) testing and integration testing.

Topic 4.2: Definition

"Testing is the process of executing a program with intention of finding errors."

Software testing is a formal process carried out by a specialized testing team in which a software unit, several integrated software units or an entire software package are examined by running the programs on a computer.

All the associated tests are performed according to approved test procedures on approved test cases.

Formal – Software test plans are part of the project's development and quality plans, scheduled in advance and often a central item in the development agreement signed between the customer and the developer.

Specialized testing team – An independent team or external consultants who specialize in testing are assigned to perform these tasks mainly in order to eliminate bias and to guarantee effective testing by trained professionals.

Running the programs – Any form of quality assurance activity that does not involve running the software, for example code inspection, cannot be considered as a test.

Approved test procedures – The testing process performed according to a test plan and testing procedures that have been approved as conforming to the SQA procedures adopted by the developing organization.

Approved test cases – The test cases to be examined are defined in full by the test plan. No omissions or additions are expected to occur during testing.

Topic 4.3: Software testing objectives

Direct objectives

- To identify and reveal as many errors as possible in the tested software.
- To bring the tested software, after correction of the identified errors and retesting, to an acceptable level of quality.
- To perform the required tests efficiently and effectively, within budgetary and scheduling limitations.

Indirect objective

- To compile a record of software errors for use in error prevention (by corrective and preventive actions).

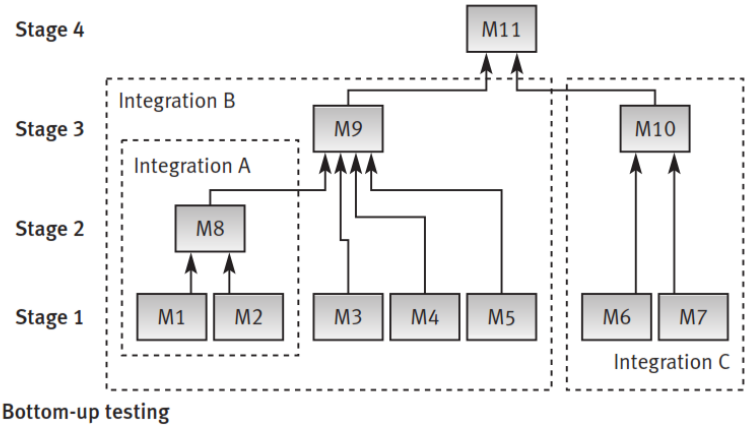
Topic 4.4: Software testing strategies

To test the software in its entirety, once the completed package is available; known as "big bang testing".

To test the software piecemeal, in modules, as they are completed (unit tests); then to test groups of tested modules integrated with newly completed modules (integration tests). This process continues until all the Package modules have been tested. Once this phase is completed, the entire package is tested as a whole (system test). This testing strategy is usually termed "incremental testing".

Incremental testing is also performed according to two basic strategies: bottom-up and top-down.

Both incremental testing strategies assume that the software package is constructed of a hierarchy of software modules.



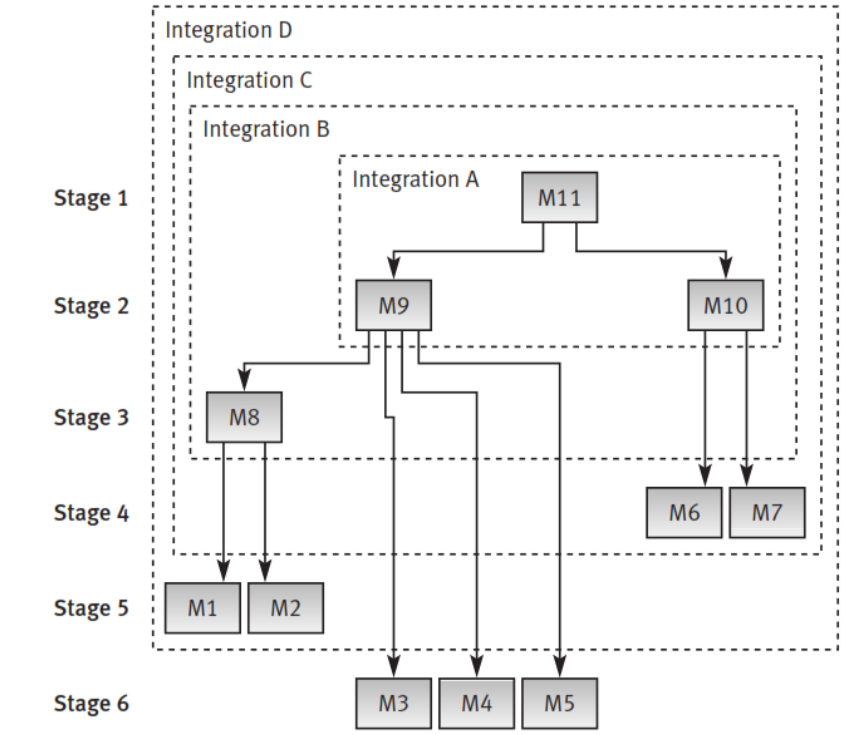
(a) Bottom-up testing

Stage 1: Unit tests of modules 1 to 7.

Stage 2: Integration test A of modules 1 and 2, developed and tested in stage 1, and integrated with module 8, developed in the current stage.

Stage 3: Two separate integration tests, B, on modules 3, 4, 5 and 8, integrated with module 9, and C, for modules 6 and 7, integrated with module 10.

Stage 4: System test is performed after B and C have been integrated with module 11, developed in the current stage.



(b) Top-down testing

Stage 1: Unit tests of module 11.

Stage 2: Integration test A of module 11 integrated with modules 9 and 10, developed in the current stage.

Stage 3: Integration test B of A integrated with module 8, developed in the current stage.

Stage 4: Integration test C of B integrated with modules 6 and 7, developed in the current stage.

Stage 5: Integration test D of C integrated with modules 1 and 2, developed in the current stage.

Stage 6: System test of D integrated with modules 3, 4 and 5, developed in the current stage.

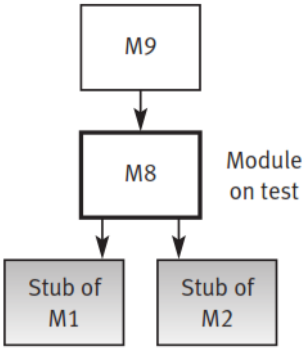
Topic 4.5: Stubs and drivers for incremental testing

Stubs and drivers are software replacement simulators required for modules not available when performing a unit or an integration test.

A stub (often termed a "dummy module") replaces an unavailable lower level module, subordinate to the module tested.

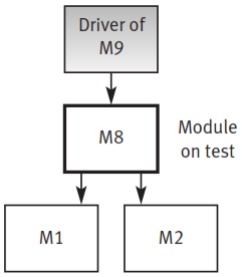
Stubs are required for topdown testing of incomplete systems. In this case, the stub provides the results of calculations the subordinate module, yet to be developed (coded), is designed to perform.

(a) Implementing top-down tests (Stage 3 testing of the example shown in Figure 9.1)



A driver is a substitute module but of the upper level module that activates the module tested. The driver is passing the test data on to the tested module and accepting the results calculated by it.

Drivers are Required in bottom-up testing until the upper level modules are developed (coded).



(b) Implementing bottom-up tests (Stage 2 testing of the example shown in Figure 9.1)

Topic 4.6: THE TESTING PROCESS

Testing is different from debugging.

- Removing errors from your programs is known as *debugging but testing aims to locate as yet undiscovered errors.*
- Starts from the requirements analysis phase and goes until the last maintenance phase.

Static testing: Requirement analysis and designing stage.

SRS is tested to check whether it is as per user requirements or not. We use techniques of code reviews, code inspections, walkthroughs, and software technical reviews (STRs) to do static testing

Dynamic testing **:** Starts when the code is ready or even a unit (or module) is ready.

It is dynamic testing as now the code is tested.

Techniques for dynamic testing like black-box, graybox, and white-box testing.

Topic 4.7: Five distinct levels of testing

a. Debug: It is defined as the successful correction of a failure.

b. Demonstrate: The process of showing that major features work with typical input.

c. Verify: The process of finding as many faults in the application under test (AUT) as possible.

d. Validate: The process of finding as many faults in requirements, design, and AUT.

e. Prevent: To avoid errors in development of requirements, design, and implementation by self-checking techniques, including “test before design.”

Topic 4.8: Popular equation of software testing

Software Testing = Software Verification + Software Validation

- Software Verification "It is the process of evaluating, reviewing, inspecting and doing desk checks of work products such as requirement specifications, design specifications and code."
- Verification means **Are we building the product right?**

Topic 4.9: software validation

- "It is defined as the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements. It involves executing the actual software. It is a computer based testing process."
- Validation means **Are we building the right product?**
- Both verification and validation (V&V) are complementary to each other.

Topic 4.10: Software test classifications

Topic 4.11: Classification according to testing concept

Black box testing:

(1) Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

(2) Testing conducted to evaluate the compliance of a system or component with specified functional requirements.

White box testing:

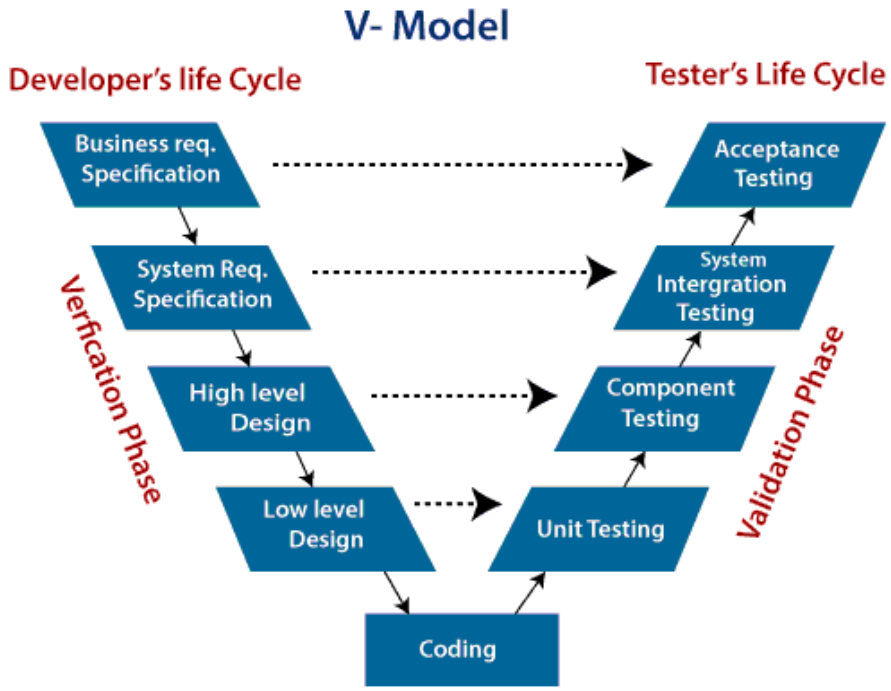
Testing that takes into account the internal mechanism of a system or component.

Topic 4.12: Classification according to requirements

Factor category	Quality requirement factor	Quality requirement sub-factor	Test classification according to requirements
Operation	1. Correctness	1.1 Accuracy and completeness of outputs, accuracy and completeness of data	1.1 Output correctness tests
		1.2 Accuracy and completeness of documentation	1.2 Documentation tests
		1.3 Availability (reaction time)	1.3 Availability (reaction time) tests
		1.4 Data processing and calculations correctness	1.4 Data processing and calculations correctness tests
		1.5 Coding and documentation standards	1.5 Software qualification tests
	2. Reliability		2. Reliability tests
	3. Efficiency		3. Stress tests (load tests, durability tests)
	4. Integrity		4. Software system security tests
	5. Usability	5.1 Training usability 5.2 Operational usability	5.1 Training usability tests 5.2 Operational usability tests

Revision	6. Maintainability 7. Flexibility 8. Testability		6. Maintainability tests 7. Flexibility tests 8. Testability tests
Transition	9. Portability 10. Reusability 11. Interoperability	11.1 Interoperability with other software 11.2 Interoperability with other equipment	9. Portability tests 10. Reusability tests 11.1 Software interoperability tests 11.2 Equipment interoperability tests

Topic 4.13: V-Model in Software Testing



V-Model in Software testing is an SDLC model where the **test execution** takes place in a hierarchical manner. The execution process makes a V-shape.

It is also called a Verification and Validation model that undertakes the testing process for every development phase.

According to the waterfall model, testing is a post-development activity.

The spiral model took one step further by breaking the product

into increments each of which can be tested separately.

V-model brings in a new perspective that different types of testing

apply at different levels.

The V-model splits testing into two parts

- DESIGN
- EXECUTION.

Verification phases on one side and the Validation phases on the other side.

Verification and Validation process is joined by coding phase in V-shape.

Test: Testing is concerned with errors, faults, failures, and incidents. A test is the act of exercising software with test cases. A test has two distinct goals—to find failures or to demonstrate correct execution.

Test case: A test case has an identity and is associated with program behavior. A test case also has a set of inputs and a list of expected outputs. The essence of software testing is to determine a set of test cases for the item to be tested.

Topic 4.14: Test case template

```
Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs
Postconditions
Execution History
Date        Result        Version        Run By
```

Test suite: A collection of test scripts or test cases that is used for validating bug fixes (or finding new bugs) within a logical or physical area of a product.

For example, an acceptance test suite contains all of the test cases that were used to verify that the software has met certain predefined acceptance criteria.

Test script: The step-by-step instructions that describe how a test case is to be executed. It may contain one or more test cases.

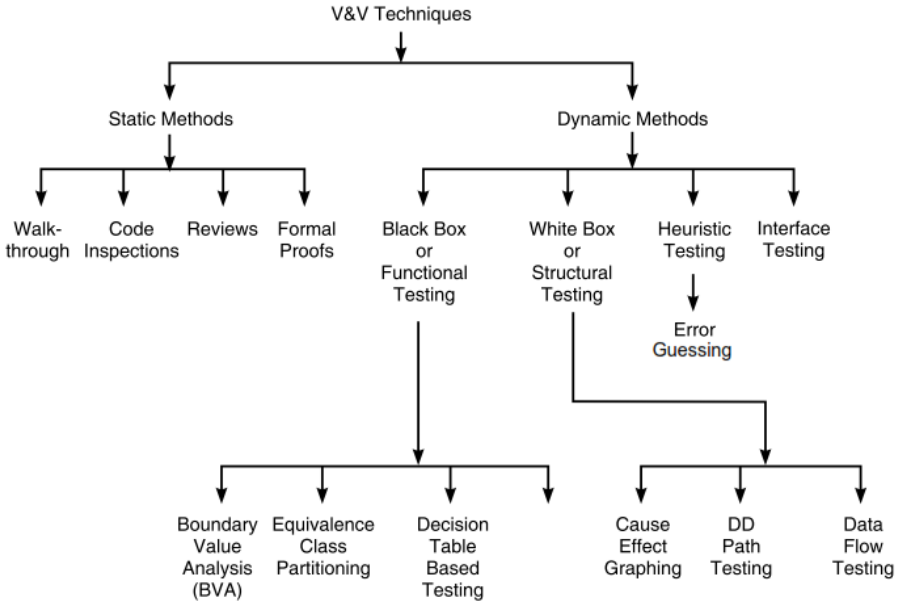
Test cases for ATM:

Preconditions: System is started.

Test case ID	Test case name	Test case description	Test steps			Test status (P/F)	Test priority
			Step	Expected result	Actual result		
Session 01	Verify Card	To verify whether the system reads a customer's ATM card.	Insert a readable card	Card is accepted; System asks for entry of PIN			High
			Insert an unreadable card	Card is ejected; System displays an error screen; System is ready to start a new session			High
	Validate PIN	To verify whether the system accepts customer's PIN	Enter valid PIN	System displays a menu of trans-action types			High
			Enter invalid PIN	Customer is asked to re-enter			High

Test case ID	Test case name	Test case description	Test steps			Test status (P/F)	Test priority
			Step	Expected result	Actual result		
			Enter incorrect PIN the first time, then correct PIN the second time	System displays a menu of transaction types.			High
			Enter incorrect PIN the first time and second time, then correct PIN the third time	System displays a menu of transaction types.			High
			Enter incorrect PIN three times	An appropriate message is displayed; Card is retained by machine; Session is terminated			High

Topic 4.15: CATEGORIZING V&V TECHNIQUES



Topic 4.16: BLACK-BOX (OR FUNCTIONAL TESTING)

The term Black-Box refers to the software which is treated as a black-box.

The system or source code is not checked at all.

It is done from the customer's viewpoint.

The test engineer engaged in black-box testing only knows the set of inputs and expected outputs and is unaware of how those inputs are transformed into outputs by the software.

Topic 4.17: BOUNDARY VALUE ANALYSIS (BVA)

It is a black-box testing technique that believes and extends the concept that the density of defect is more towards the boundaries. This is done for the following reasons:

i. Programmers usually are not able to decide whether they have to use <= operator or < operator when trying to make comparisons.

ii. Different terminating conditions of for-loops, while loops, and repeat loops may cause defects to move around the boundary conditions.

iii. The requirements themselves may not be clearly understood, especially around the boundaries, thus causing even the correctly coded program to not perform the correct way.

The basic idea of BVA is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and at their maximum.

{min, min+, nom, max–, max}

- BVA is based upon a critical assumption that is known as single fault assumption theory.
- When more than one variable for the same application is checked then one can use a single fault assumption.
- Holding all but one variable to the extreme value and allowing the remaining variable to take the extreme value. For n variable to be checked:

- ***Maximum of 4n+1 test cases***

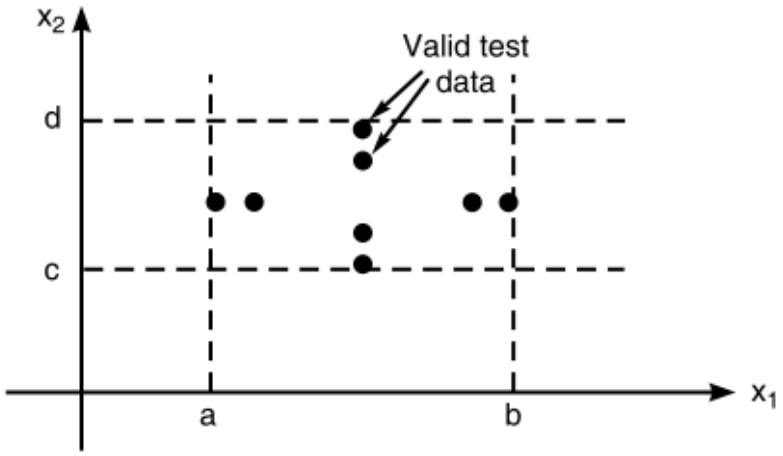


FIGURE 3.1 BVA Test Cases.

Problem: Consider a Program for determining the Previous Date.

Input: *Day, Month, Year with valid ranges as-*

$1 \leq Month \leq 12$
$1 \leq Day \leq 31$
$1900 \leq Year \leq 2000$

Design Boundary Value Test Cases.

Solution:

1) Year as a Single Fault Assumption

Test Cases	Month	Day	Year	Output
1	6	15	1900	14 June 1900
2	6	15	1901	14 June 1901
3	6	15	1960	14 June 1960
4	6	15	1999	14 June 1999
5	6	15	2000	14 June 2000

Day as Single Fault Assumption

Test Case	Month	Day	Year	Output
6	6	1	1960	31 May 1960
7	6	2	1960	1 June 1960
8	6	30	1960	29 June 1960
9	6	31	1960	Invalid day

Month as Single Fault Assumption

Test Case	Month	Day	Year	Output
10	1	15	1960	14 Jan 1960
11	2	15	1960	14 Feb 1960
12	11	15	1960	14 Nov 1960
13	12	15	1960	14 Dec 1960

For the n variable to be checked Maximum of 4n + 1 test case will be required.

Therefore, for n = 3, the maximum test cases are

$$4 \times 3 + 1 = 13$$

Consider a system that accepts ages from 18 to 56.

Boundary Value Analysis(Age accepts 18 to 56)		
Invalid (min-1)	Valid (min, min + 1, nominal, max – 1, max)	Invalid (max + 1)
17	18, 19, 37, 55, 56	57

Valid Test cases: Valid test cases for the above can be any value entered greater than 17 and less than 57.

Enter the value- 18.

Enter the value- 19.

Enter the value- 37.

Enter the value- 55.

Enter the value- 56.

Invalid Testcases: When any value less than 18 and greater than 56 is entered.

Enter the value- 17.

Enter the value- 57.

Topic 4.18: EQUIVALENCE CLASS TESTING

The use of equivalence classes as the basis for functional testing has two motivations:

a. We want exhaustive testing

b. We want to avoid redundancy

This is not handled by the BVA technique as we can see massive redundancy in the tables of test cases.

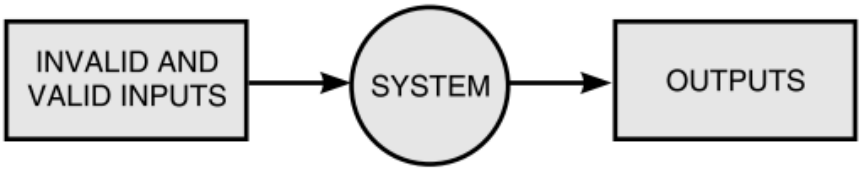


FIGURE 3.4 Equivalence Class Partitioning.

Topic 4.19: Process

- The input and the output domain **is divided into a** finite number of equivalence classes.
- select one representative of each class and test our program against it.
- It is assumed by the tester that if one representative from a class is able to detect error then why should he consider other cases.
- if this single representative test case did not detect any error then we assume that no other test case of this class can detect error.
- we consider both valid and invalid input domains.
- The key and the craftsmanship lies in the choice of the equivalence relation that determines the classes.
- The equivalence class testing can be categorized into four different types.
- **Weak Normal Equivalence Class Testing:** In this first type of equivalence class testing, one variable from each equivalence class is tested by the team. Moreover, the values are identified in a systematic manner. Weak normal equivalence class testing is also known as **single fault assumption**.
- **Strong Normal Equivalence Class Testing:** Termed as **multiple fault assumption**, in strong normal equivalence class testing the team selects test cases from each element of the Cartesian product of the equivalence. This ensures the notion of completeness in testing, as it

covers all equivalence classes and offers the team one of each possible combinations of inputs.

- **Weak Robust Equivalence Class Testing:** Like weak normal equivalence, weak robust testing too tests one variable from each equivalence class. However, unlike the former method, it is also focused on testing test cases for invalid values.
- **Strong Robust Equivalence Class Testing:** Another type of equivalence class testing, strong robust testing produces test cases for all valid and invalid elements of the product of the equivalence class. However, it is incapable of reducing the redundancy in testing.

Topic 4.20: White Box Testing

- White-box testing is a way of testing the external functionality of the code by examining and testing the program code that realizes the external functionality.
- White-box testing is used to test the program code, code structure, and the internal design flow.

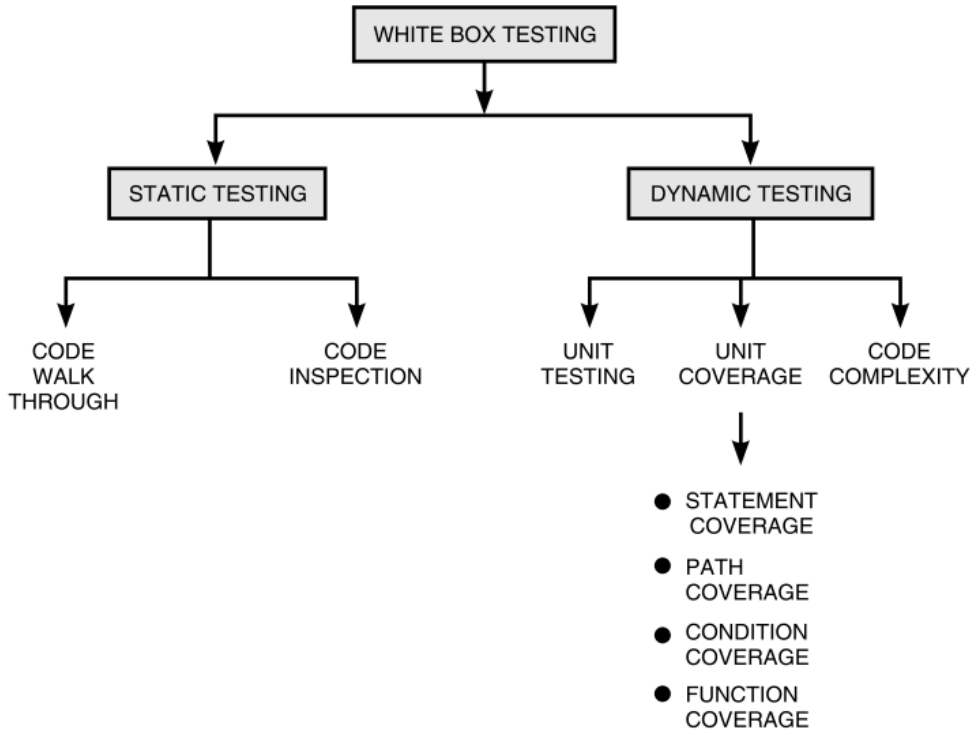


FIGURE 4.1 Classification of White-Box Testing.

Topic 4.21: CODE COVERAGE TESTING

- Designing and executing test cases and finding out the percentage of code that is covered by testing.
- The percentage of code covered by a test is found by adopting a technique called the instrumentation of code.
- *STATEMENT COVERAGE*
- *PATH COVERAGE*
- *CONDITION COVERAGE*

- *FUNCTION COVERAGE*

5. Software Quality Assurance and TestingModule 5

Topic 5.1 Test levels and types

Three levels of testing:

1. Unit testing

2. Integration testing

3. System testing

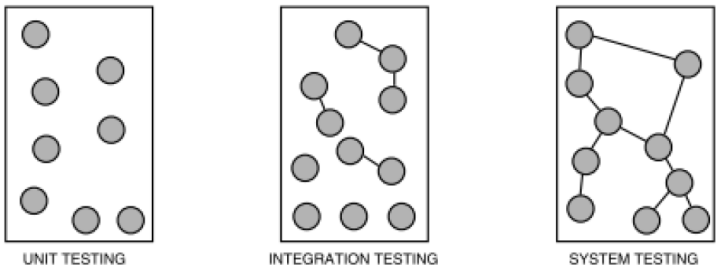
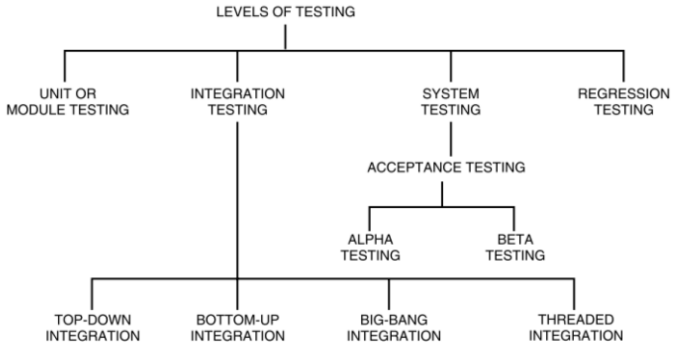


FIGURE 7.1 Levels of Testing.



Topic 5.2 Unit (or Module) Testing

- **Unit (or module) testing** is the process of isolating a module (an atomic unit of software) and running it independently from the rest of the software system using prepared test cases. The actual results are then compared to the expected results based on the module's specification and design.
  It is a **white-box testing technique**.

Topic 5.3 Importance of Unit Testing:

1. Testing individual modules simplifies the process and increases efficiency.
2. More exhaustive test coverage can be achieved.
3. Interface errors are identified and eliminated early.

Topic 5.4 Integration Testing

- A system is composed of multiple components or modules, which can include both hardware and software. **Integration** refers to the set of interactions among these components.
- **Integration Testing** involves testing the interaction between modules and their interactions with external systems.
  The architecture and design of the system provide details about these interactions, forming the basis for the integration testing process.

Topic 5.5 Classification of integration testing

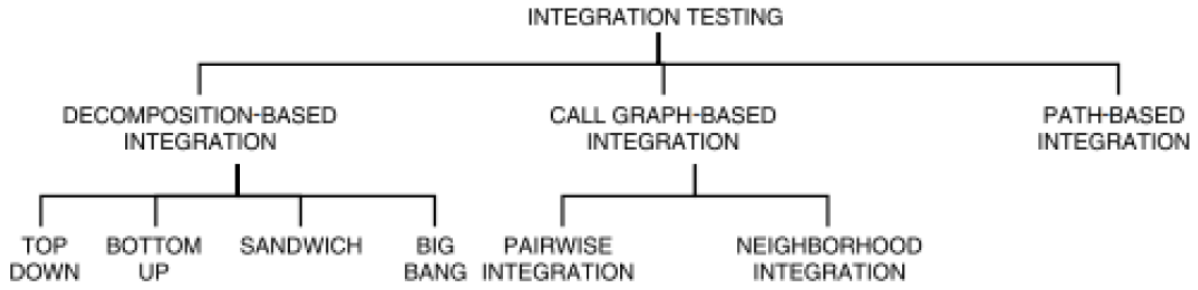


FIGURE 7.4

Topic 5.6 Decomposition-Based Integration

- **Definition**: Decomposition-based integration involves the functional decomposition of the system to be tested, represented as a tree or in textual form. The goal is to test the interfaces among independently tested units.

Topic 5.7 Types of Decomposition-Based Techniques:

1. Top-Down Integration Approach

- Starts with the main program, the root of the tree.
- Lower-level units that are called by the main program are replaced by *stubs*.
- **Stub**: A piece of throw-away code that mimics a called unit.

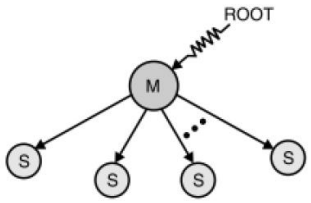


FIGURE 7.5 Stubs.

2. Bottom-Up Integration Approach

- A mirror image of the top-down approach.
- **Driver modules** are used instead of stubs to emulate higher-level units.
- Testing begins with the leaves of the decomposition tree.
- Requires less throw-away code compared to stubs.

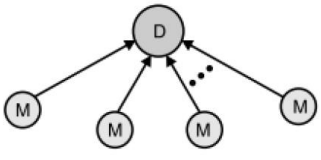


FIGURE 7.6 Drivers.

Topic 5.8 Sandwich Integration Approach

- It is a combination of top-down and bottom-up integration, bidirectional integration
- There will be less stub and driver development effort.

- Combination of the top-down and bottom-up integration approaches, it is also called bidirectional integration.
- It is performed initially with the use of stubs and drivers.
- Focuses on those components which need focus and are new.

Topic 5.9 Big-bang Integration

- Instead of integrating component by component and testing, this approach waits until all the components arrive, and one round of integration testing is done. This is known as big-bang integration. It reduces testing effort and removes duplication in testing for the multi-step component integrations. Big-bang integration is ideal for a product where the interfaces are stable with fewer number of defects.

7.2.2.6. GUIDELINES TO CHOOSE INTEGRATION METHOD AND CONCLUSIONS

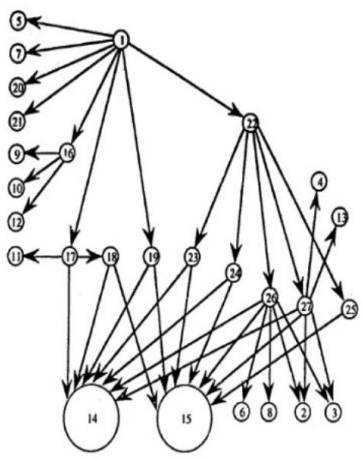
S. No.	Factors	Suggested method
1.	Clear requirements and design.	Top-down approach.
2.	Dynamically changing requirements, design, and architecture.	Bottom-up approach.
3.	Changing architecture and stable design.	Sandwich (or bi-directional) approach.
4.	Limited changes to existing architecture with less impact.	Big-bang method.
5.	Combination of above.	Select any one after proper analysis.

Topic 5.10 Call Graph-based Integration

Structural Testing:
A call graph is a directed graph.
Since a call graph is a directed graph, we can use it as a program graph.
Pairwise Integration
Neighborhood Integration

Table 13.1 SATM Units and Abbreviated Names

Unit Number	Level Number	Unit Name
1	1	SATM System
A	1.1	Device Sense & Control
D	1.1.1	Door Sense & Control
2	1.1.1.1	Get Door Status
3	1.1.1.2	Control Door
4	1.1.1.3	Dispense Cash
E	1.1.2	Slot Sense & Control
5	1.1.2.1	WatchCardSlot
6	1.1.2.2	Get Deposit Slot Status
7	1.1.2.3	Control Card Roller
8	1.1.2.3	Control Envelope Roller
9	1.1.2.5	Read Card Strip
10	1.2	Central Bank Comm.
11	1.2.1	Get PIN for PAN
12	1.2.2	Get Account Status
13	1.2.3	Post Daily Transactions
B	1.3	Terminal Sense & Control
14	1.3.1	Screen Driver
15	1.3.2	Key Sensor
C	1.4	Manage Session
16	1.4.1	Validate Card
17	1.4.2	Validate PIN
18	1.4.2.1	GetPIN
F	1.4.3	Close Session
19	1.4.3.1	New Transaction Request
20	1.4.3.2	Print Receipt
21	1.4.3.3	Post Transaction Local
22	1.4.4	Manage Transaction
23	1.4.4.1	Get Transaction Type
24	1.4.4.2	Get Account Type
25	1.4.4.3	Report Balance
26	1.4.4.4	Process Deposit
27	1.4.4.5	Process Withdrawal



Topic 5.11 Pairwise Integration

- The main idea behind pairwise integration is to eliminate the stub/driver development effort.
- Instead of developing stubs and drivers, why not use the actual code?

- we restrict a session to only a pair of units in the call graph.
- The end result is that we have one integration test session for each edge in the call graph.
- This is not much of a reduction in sessions from either topdown or bottom-up but it is a drastic reduction in stub/driver development.

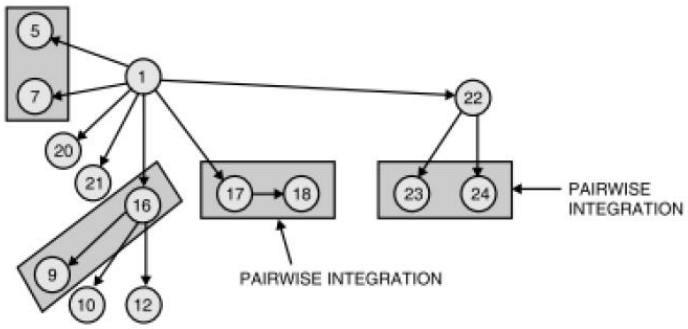


FIGURE 7.7 Pairwise Integration.

Topic 5.12 Neighborhood Integration

- The neighborhood of a node in a graph is the set of nodes that are one edge away from the given node.

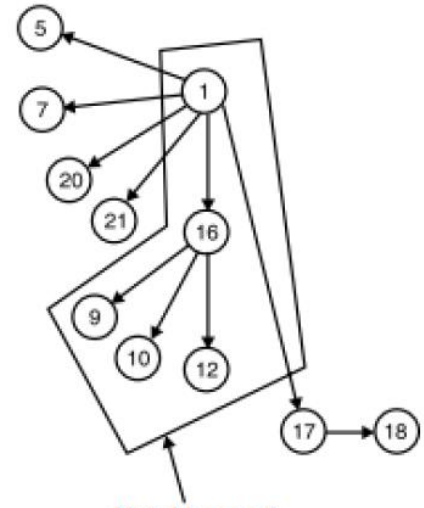


FIGURE 7.8 Neighborhood Integration.

Topic 5.13 Path-based Integration

- The combination of structural and functional testing is highly desirable at the unit level, and it would be nice to have a similar capability for integration and system testing.
- "Instead of testing interfaces among separately developed and tested units, we focus on interactions among these units."
- Here, co-functioning might be a good term.
- Interfaces are structural, whereas interaction is behavioral.

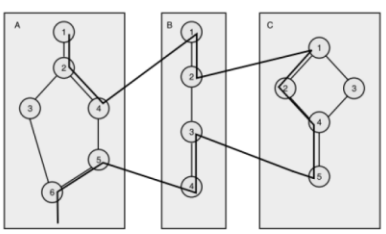


FIGURE 7.9 MM-Path Across Three Units (A, B, and C).

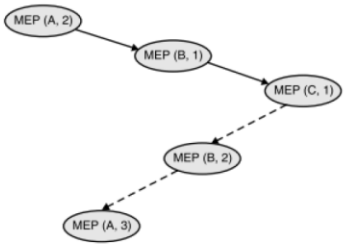


FIGURE 7.10 MM-Path Graph Derived from Figure 7.9.

Topic 5.14 SYSTEM TESTING

- The testing that is conducted on the complete integrated products and solutions to evaluate system compliance with specified requirements on functional and non-functional aspects is called system testing. It is done after unit, component, and integration testing phases.
- System testing is done to:
  - Provide an independent perspective in testing as the team becomes more quality-centric.
  - Bring in customer perspective in testing.
  - Provide a "fresh pair of eyes" to discover defects not found earlier by testing.
  - Test product behavior in a holistic, complete, and realistic environment.
  - Test both functional and non-functional aspects of the product.
  - Build confidence in the product.
  - Analyze and reduce the risk of releasing the product.
  - Ensure all requirements are met and ready the product for acceptance testing.
- An independent test team normally performs system testing.
- This independent test team is different from the team responsible for component and integration testing.
- The system test team generally reports to a manager other than the product manager to avoid conflicts and provide freedom to individuals during system testing.
- Testing the product with an independent perspective and combining that with the customer's perspective makes system testing unique, different, and effective.

Functional testing	Non functional testing
1. It involves the product's functionality.	1. It involves the product's quality factors.
2. Failures, here, occur due to code.	2. Failures occur due to either architecture, design, or due to code.
3. It is done during unit, component, integration, and system testing phase.	3. It is done in our system testing phase.
4. To do this type of testing only domain of the product is required.	4. To do this type of testing, we need domain, design, architecture, and product's knowledge.
5. Configuration remains same for a test suite.	5. Test configuration is different for each test suite.

Topic 5.15 Module 5 Test Execution Process

Topic 5.16 Test Plan

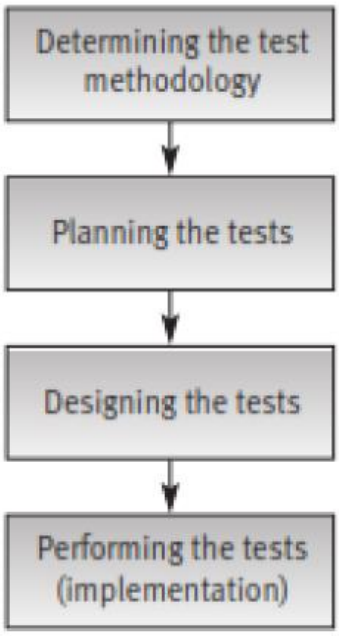
The primary purpose of a test plan is to define the scope, approach, resources, and schedule for testing activities. It provides a systematic approach to ensure that the software meets specified requirements and quality standards.

Topic 5.17 Key components of a test plan

- Objectives,
- Scope,
- Test Items,
- Test Environment,
- Testing Strategy,
- Test Schedule,
- Resource Requirements,
- Risk Management
- Test Deliverables.

Topic 5.18 The Testing Process

➢ Planning, design, and performance of testing are carried out throughout the software development process. These activities are divided into phases, beginning in the design stage and ending when the software is installed at the customer's site.



Topic 5.19 Determining the test methodology phase

➢ The main issues that testing methodology has to contend with are:

➢ The appropriate required software quality standard

- The software testing strategy.

Topic 5.20 Determining the Appropriate Software Quality Standard

The level of quality standard selected for a project depends mainly on the characteristics of the software's application.
Example 1: A software package for a hospital patient bed monitor requires the highest software quality standard, considering the possibly severe consequences of software failure.

Determining the Software Testing Strategy
The issues that have to be decided include:

The testing strategy: Should a big bang or incremental testing strategy be adopted? If incremental testing is preferable, should testing be performed bottom-up or top-down?
Which parts of the testing plan should be performed according to the white-box testing model?
Which parts of the testing plan should be performed according to the automated testing model?

Topic 5.21 Planning the Tests

The tests to be planned include:

- Unit tests
- Integration tests
- System tests

Consider the following issues before initiating a specific test plan:

- What to test?
- Which sources to use for test cases?
- Who is to perform the tests?
- Where to perform the tests?
- When to terminate the tests?

Topic 5.22 Test Planning Documentation

The planning stage of the software system tests is commonly documented in a "Software Test Plan" (STP).

Frame 10.2 **The software test plan (STP) – template**

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)
1.2 The documents that provide the basis for the planned tests (name and version for each document)

2 Testing environment

2.1 Testing sites
2.2 Required hardware and firmware configuration
2.3 Participating organizations
2.4 Manpower requirements
2.5 Preparation and training required of the test team

3 Test details (for each test)

3.1 Test identification
3.2 Test objective
3.3 Cross-reference to the relevant design document and the requirement document
3.4 Test class
3.5 Test level (unit, integration or system tests)
3.6 Test case requirements
3.7 Special requirements (e.g., measurements of response times, security requirements)
3.8 Data to be recorded

4 Test schedule (for each test or test group) including time estimates for the following:

4.1 Preparation
4.2 Testing
4.3 Error correction
4.4 Regression tests

Topic 5.23 Test design

The products of the test design stage are:

Detailed design and procedures for each test

Test case database/file.

The test design is carried out on the basis of the software test plan as documented by STP.

The test procedures and the test case database/file may be documented in a “software test procedure” document and “test case file” document or in a single document called the “software test description” (STD).

Frame 10.3 **Software test descriptions (STD) – template**

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)
1.2 The documents providing the basis for the designed tests (name and version for each document)

2 Test environment (for each test)

2.1 Test identification (the test details are documented in the STP)
2.2 Detailed description of the operating system and hardware configuration and the required switch settings for the tests
2.3 Instructions for software loading

3 Testing process

3.1 Instructions for input, detailing every step of the input process
3.2 Data to be recorded during the tests

4 Test cases (for each case)

4.1 Test case identification details
4.2 Input data and system settings
4.3 Expected intermediate results (if applicable)
4.4 Expected results (numerical, message, activation of equipment, etc.)

5 Actions to be taken in case of program failure/cessation

6 Procedures to be applied according to the test results summary

Topic 5.24 Test Implementation

The testing implementation phase activities consist of a series of tests, corrections of detected errors, and re-tests (regression tests). Testing is culminated when the re-test results satisfy the developers. The tests are carried out by running the test cases according to the test procedures.

Documentation of the test procedures and the test case database/file comprises the "Software Test Description" (STD).

Re-testing (also termed "regression testing") is conducted to verify that the errors detected in the previous test runs have been properly corrected, and that no new errors have entered as a result of faulty corrections.

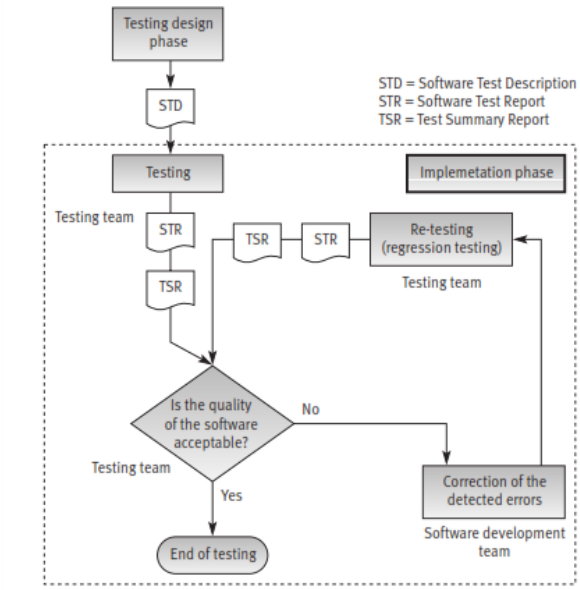


Figure 10.2: Implementation phase activities

The summary of the set of tests planned for a software package (or software development project) is documented in the "Test Summary Report" (TSR).

Frame 10.4 **Software test report (STR) – template**

1 Test identification, site, schedule and participation

1.1 The tested software identification (name, version and revision)
1.2 The documents providing the basis for the tests (name and version for each document)
1.3 Test site
1.4 Initiation and concluding times for each testing session
1.5 Test team members
1.6 Other participants
1.7 Hours invested in performing the tests

2 Test environment

2.1 Hardware and firmware configurations
2.2 Preparations and training prior to testing

3 Test results

3.1 Test identification
3.2 Test case results (for each test case individually)
  3.2.1 Test case identification
  3.2.2 Tester identification
  3.2.3 Results: OK / failed
  3.2.4 If failed: detailed description of the results/problems

4 Summary tables for total number of errors, their distribution and types

4.1 Summary of current tests
4.2 Comparison with previous results (for regression test summaries)

5 Special events and testers' proposals

5.1 Special events and unpredicted responses of the software during testing
5.2 Problems encountered during testing
5.3 Proposals for changes in the test environment, including test preparations
5.4 Proposals for changes or corrections in test procedures and test case files

Topic 5.25 Test Case Design

Topic 5.26 Test Case Data Components

A test case is a documented set of the data inputs and operating conditions required to run a test item, together with the expected results of the run. The tester is expected to run the program for the test item according to the test case documentation, and then compare the actual results with the expected results noted in the documents.
If the obtained results completely agree with the expected results, no error is present or at least has been identified. When some or all of the results do not agree with the expected results, a potential error is identified.

Example
Consider the following test cases for the basic annual municipal property tax on apartments. The basic municipal property tax (before discounts to special groups of city dwellers) is based on the following parameters:

S, the size of the apartment (in square yards)
N, the number of persons living in the apartment
A, B or C, the suburb's socio-economic classification.

The municipal property tax (MPT) is calculated as follows:

For class A suburbs: $MPT = (100 \times S) / (N + 8)$
For class B suburbs $MPT = (80 \times S) / (N + 8)$
For class C suburbs $MPT = (50 \times S) / (N + 8)$

The following are three test cases for the software module used to calculate the basic municipal property tax on apartments:

	Test case 1	Test case 2	Test case 3
Size of apartment – (square yards), *S*	250	180	98
Suburb class	A	B	C
No. of persons in the household, *N*	2	4	6
Expected result: municipal property tax (MPT)	**$2500**	**$1200**	**$350**

Topic 5.27 Automated Testing

Automated testing represents an additional step in the integration of computerized tools into the process of software development. These tools have joined computer-aided software engineering (CASE) tools in performing a growing share of software analysis and design tasks.

Factors have motivated the development of automated testing tools:

Anticipated cost savings
Shortened test duration
Heightened thoroughness of the tests performed
Improvement of test accuracy
Improvement of result reporting, as well as statistical processing and subsequent reporting

Topic 5.28 The Process of Automated Testing

Automated software testing requires test planning, test design, test case preparation, test performance, test log and report preparation, re-testing after correction of detected errors (regression tests), and final test log and report preparation, including comparison reports. The last two activities may be repeated several times.

Topic 5.29 Types of Automated Tests

- **Code Auditing:** The computerized code auditor checks the compliance of code to specified standards and procedures of coding. The auditor's report includes a list of the deviations from the standards and a statistical summary of the findings.
- **Coverage Monitoring:** Coverage monitors produce reports about the line coverage achieved when implementing a given test case file. The monitor's output includes the percentage of lines covered by the test cases as well as listings of uncovered lines. These features make coverage monitoring a vital tool for white-box tests.
- **Functional Tests:** Automated functional tests often replace manual black-box correctness tests. Prior to performing these tests, the test cases are recorded in the test case database. The tests are then carried out by executing the test cases through the test program. The test results documentation includes listings of the errors identified, in addition to a variety of summaries and statistics as demanded by the testers' specifications.
- **Test Management:** Testing involves many participants who are occupied in actually carrying out the tests and correcting the detected errors. Testing typically monitors the performance of every item on long lists of test case files. This workload makes timetable follow-up important to management. Computerized test management supports these and other testing management goals.

Topic 5.30 Advantages of Automated Tests

1. Accuracy and completeness of performance.
2. Accuracy of results log and summary reports.
3. Comprehensiveness of information.
4. Few manpower resources required to perform tests.
5. Shorter duration of testing.
6. Performance of complete regression tests.
7. Performance of test classes beyond the scope of manual testing.

Topic 5.31 Disadvantages of Automated Testing

1. High investments required in package purchasing and training.
2. High package development investment costs.
3. High manpower requirements for test preparation.
4. Considerable testing areas left uncovered.

Frame 10.7 **Automated software testing: advantages and disadvantages**

Advantages	Disadvantages
1. Accuracy and completeness of performance	1. High investments required in package purchasing and training
2. Accuracy of results log and summary reports	2. High package development investment costs
3. Comprehensive information	3. High manpower resources for test preparation
4. Few manpower resources for test execution	4. Considerable testing areas left uncovered
5. Shorter testing periods	
6. Performance of complete regression tests	
7. Performance of test classes beyond the scope of manual testing	

Topic 5.32 Alpha and Beta Site Testing Programs

Alpha site and beta site tests are employed to obtain comments about quality from the package's potential users. They are additional commonly used tools to identify software design and code errors in software packages in commercial off-the-shelf (COTS) sales.

- **Alpha Site Tests:** "Alpha site tests" are tests of a new software package that are performed at the developer's site.
- **Beta Site Tests:** Once an advanced version of the software package is available, the developer offers it free of charge to one or more potential users. The users install the package at their sites (usually called the "beta sites"), with the understanding that they will inform the developer of all the errors revealed during trials or regular usage.

Topic 5.33 Regression Testing Technique

Regression testing is used to confirm that fixed bugs have been fixed and that new bugs have not been introduced. How many cycles of regression testing are required will depend upon the project size. Cycles of regression testing may be performed once per milestone or once per build. Regression tests can be automated.

The Regression–Test Process

The regression-test process is shown in Figure 6.6.

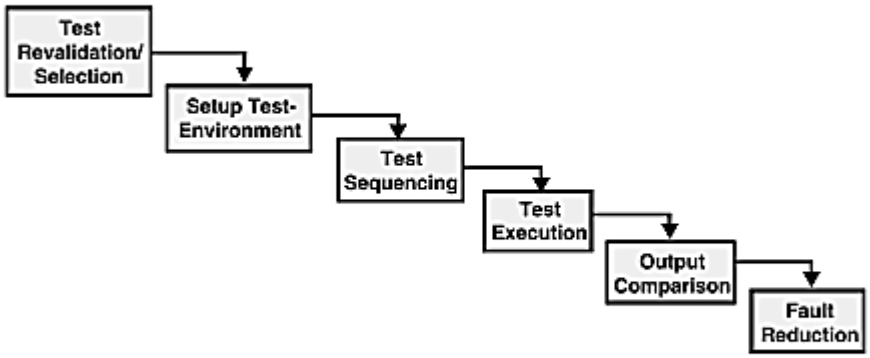


FIGURE 6.6

This process assumes that P′ (modified is program) available for regression testing. There is a long series of tasks that lead to P′ from P.

6. CS 9 Compatibility

Topic 6.1 Quality Audits and Project Assessments

- Humphrey (2005) [HUM 05] : years of data from thousands of software engineers showing that they unintentionally inject 100 defects per thousand lines of code.
- He also indicates that commercial software typically includes from one to ten errors per thousand lines of code [HUM 02].
- These errors are like hidden time bombs that will explode when certain conditions are met.
- Put practices in place to identify and correct these errors at each stage of the development and maintenance cycle.

Topic 6.2 Cost of quality

$$\begin{aligned}\text{Quality costs} = {} & \text{Prevention costs} \\ & + \text{Appraisal or evaluation costs} \\ & + \text{Internal and external failure costs} \\ & + \text{Warranty claims and loss of reputation costs}\end{aligned}$$

Cost of quality

The detection cost is the cost of verification or evaluation of a product or service during the various stages of the development process.

- One of the detection techniques is conducting reviews.
- Another technique is conducting tests.
- Quality of a software product begins in the first stage of the development process (defining requirements and specifications).
- Reviews will detect and correct errors in the early phase of development
- Tests will only be used when the code is available.
- we should not wait for the testing phase to begin to look for errors.
- Reviews range from informal to formal

Topic 6.3 Informal reviews

– There is no documented process to describe reviews and they are carried out in many ways by different people in the organization;

– Participants' roles are not defined;

– Reviews have no objective, such as fault detection rate;

– They are not planned, they are improvised;

– Measures, such as the number of defects, are not collected;

– The effectiveness of reviews is not monitored by management;

– There is no standard that describes them;

– No checklist is used to identify defects.

Review

A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

ISO 24765 [ISO 17a]

A process or meeting during which a software product, set of software products, or a software process is presented to project personnel, managers, users, customers, user representatives, auditors, or other interested parties for examination, comment, or approval.

IEEE 1028 [IEE 08b]

- IEEE 1028 standard [IEE 08b]: the walk-through and the inspection.
- The personal review and the desk-check.
- Peer reviews are product activity reviews conducted by colleagues during development, maintenance, or operations in order to present alternatives, identify errors, or discuss solutions.

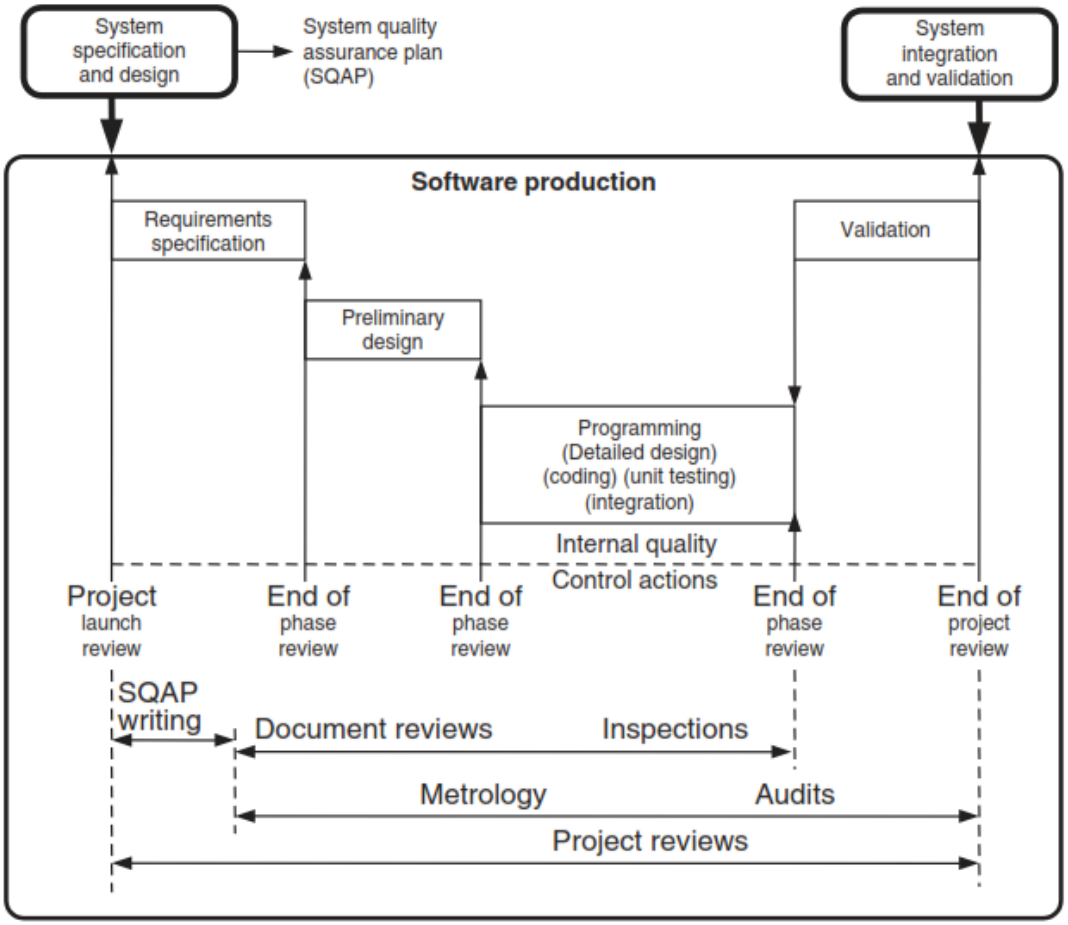


Figure 5.1 Types of reviews used during the software development cycle [CEG 90] (© 1990 – ALSTOM Transport SA).

- Identify defects
- Assess / measure the quality of a document (e.g., the number of defects per page)
- Reduce the number of defects by correcting the defects identified
- Reduce the cost of preparing future documents (i.e., by learning the type of defects each developer makes, it is possible to reduce the number of defects injected in a new document)
- Estimate the effectiveness of a process (e.g., the percentage of fault detection)
- Estimate the efficiency of a process (e.g., the cost of detection or correction of a defect)
- Estimate the number of residual defects (i.e., defects not detected when software is delivered to customer)
- Reduce the cost of tests
- Reduce delays in delivery
- Determine the criteria for triggering a process
- Determine the completion criteria of a process
- Estimate the impacts (e.g., cost) of continuing with current plans, e.g. cost of delay, recovery, maintenance, or fault remediation
- Estimate the productivity and quality of organizations, teams and individuals
- Teach personnel to follow the standards and use templates
- Teach personnel how to follow technical standards
- Motivate personnel to use the organization's documentation standards
- Prompt a group to take responsibility for decisions
- Stimulate creativity and the contribution of the best ideas with reviews
- Provide rapid feedback before investing too much time and effort in certain activities
- Discuss alternatives
- Propose solutions, improvements
- Train staff
- Transfer knowledge (e.g., from a senior developer to a junior)
- Present and discuss the progress of a project
- Identify differences in specifications and standards
- Provide management with confirmation of the technical state of the project
- Determine the status of plans and schedules
- Confirm requirements and their assignment in the system to be developed

Figure 5.2 Objectives of a review.
Source: Adapted from Gilb (2008) [GIL 08] and IEEE 1028 [IEE 08b].

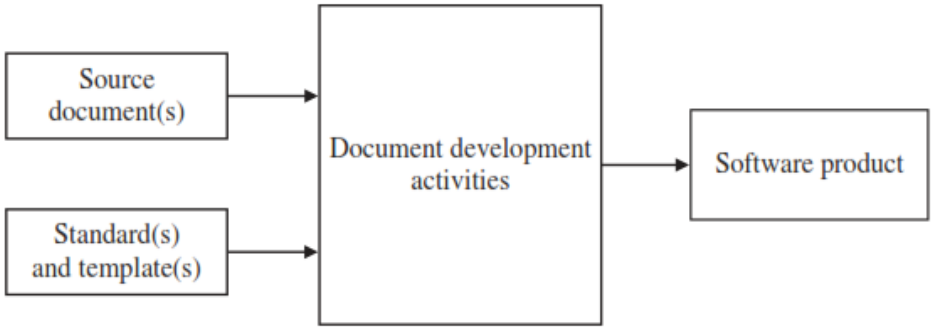


Figure 5.3 Process of developing a document.

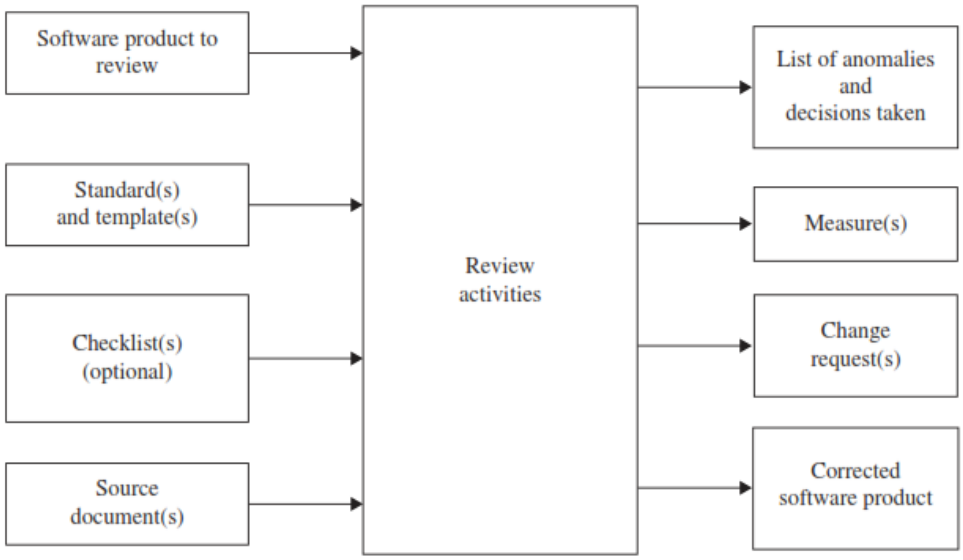


Figure 5.4 Review process.

Topic 6.4 PERSONAL REVIEW AND DESK-CHECK REVIEW

- Inexpensive and very easy to perform.
- Personal reviews do not require the participation of additional reviewers,
- Desk-check reviews require at least one other person to review the work of the developer of a software product.

Personal Review

Done by the person reviewing his own software product in order to find and fix the most defects possible.

- Principles of a personal review

– find and correct all defects in the software product;

– use a checklist produced from your personal data, if possible, using the type of defects that you are already aware of;

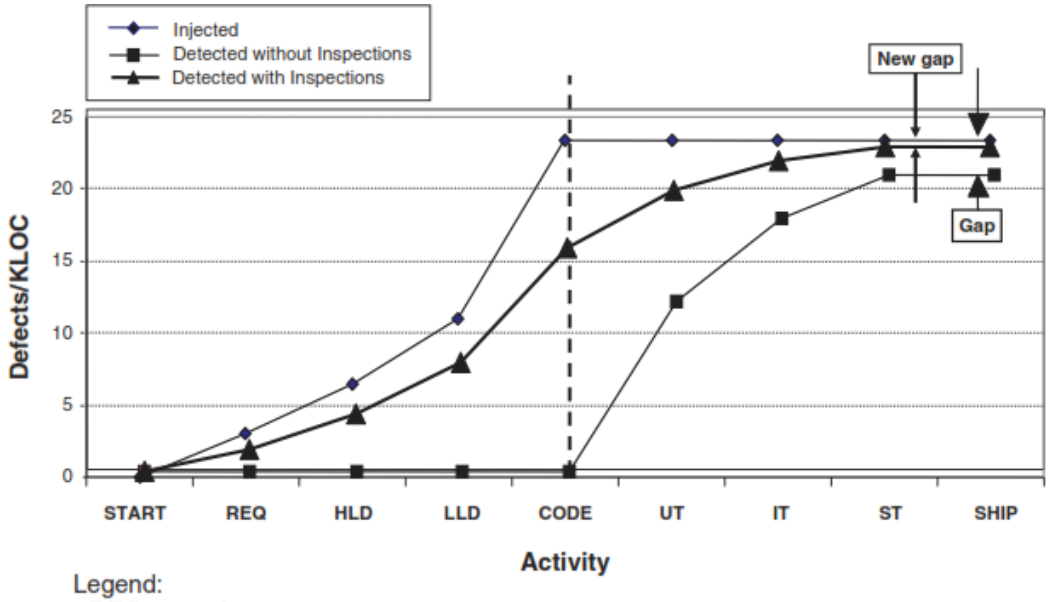
– follow a structured review process;

– use measures in your review;

– use data to improve your review;

– use data to determine where and why defects were introduced and then change

your process to prevent similar defects in the future.



Legend:
- REQ = requirement
- HLD = high-level architecture design
- LLD = detailed design
- CODE = coding and debugging
- UT = unit testing
- IT = integration testing
- ST = system testing
- SHIP = delivery to customer
- KLOC = thousand lines of code

Figure 5.5 Error detection during the software development life cycle [RAD 02].

Checklist

A checklist is used as a memory aid. A checklist includes a list of criteria to verify the quality of a product. It also ensures consistency and completeness in the development of a task. An example of a checklist is a list that facilitates the classification of a defect in a software product (e.g., an oversight, a contradiction, an omission).

Topic 6.5 practices - to develop an effective and efficient

personal review

– pause between the development of a software product and its review;

– examine products in hard copy rather than electronically;

– check each item on the checklist once completed;

– update the checklists periodically to adjust to your personal data;

– build and use a different checklist for each software product;

– verify complex or critical elements with an in depth analysis.

ENTRY CRITERIA
- None

INPUT
- Software product to review

ACTIVITIES
1. Print:
- Checklist for the software product to be reviewed
- Standard (if applicable)
- Software product to review

2. Review the software product, using the first item on the checklist and cross this item off when the review of the software product is completed
3. Continue review of the software product using the next item on the checklist and repeat until all the items in the list have been checked
4. Correct any defects identified
5. Check that each correction did not create other defects.

EXIT CRITERIA
- Corrected software product

OUTPUT
- Corrected software product

MEASURE
- Effort used to review and correct the software product measured in person-hours with an accuracy of +/–15 minutes.

Figure 5.6 Personal review process.
Source: Adapted from Pomeroy-Huff et al. (2009) [POM 09].

Topic 6.6 Desk-Check Reviews

- A type of peer review that is not described in standards is the desk-check review (Pass around) .

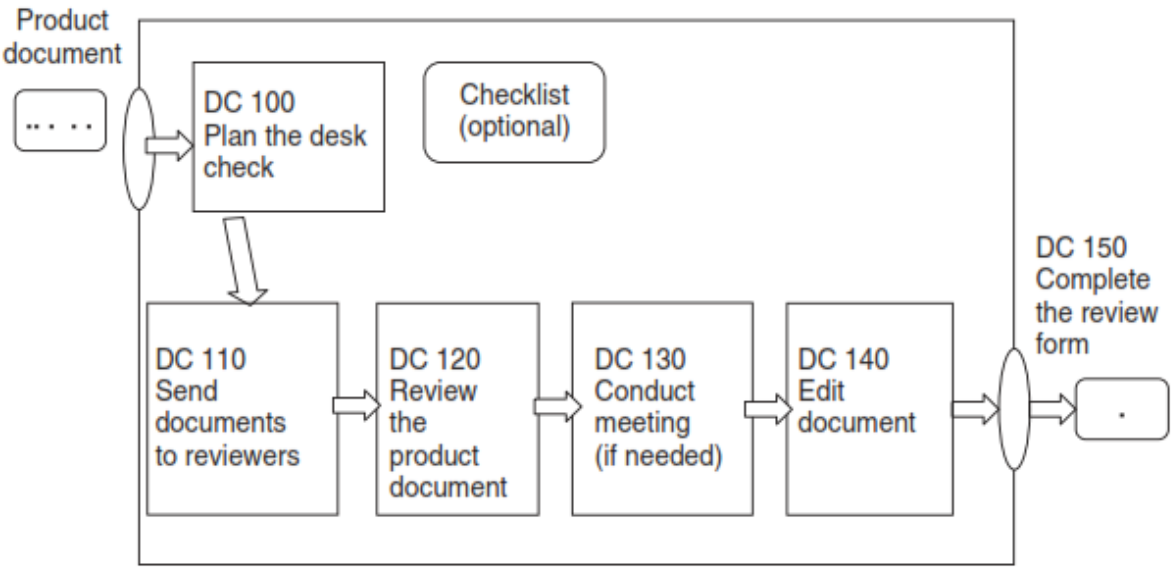


Figure 5.7 Desk-check review.

There are six steps.

Initially, the author plans the review by identifying the reviewer(s) and a checklist.

A checklist is an important element of a review as it enables the reviewer to focus on only one criterion at a time.

A checklist is a reflection of the experience of the organization.

Then, individuals review the software product document and note comments on the review form provided by the author.

When completed, the review form can be used as "evidence" during an audit.

Topic 6.7 Important features of checklists:

– each checklist is designed for a specific type of document (e.g., project plan, specification document);

– each item of a checklist targets a single verification criteria;

– each item of a checklist is designed to detect major errors. Minor errors, such as misspellings, should not be part of a checklist;

– each checklist should not exceed one page, otherwise it will be more difficult to use by the reviewers;

– each checklist should be updated to increase efficiency;

– each checklist includes a version number and a revision date.

The following text box presents a generic checklist, that is, a checklist that can be used for almost any type of document to be reviewed (e.g., project plan, architecture).

For each type of software product (e.g., requirements or design), a specific checklist will be used.

Generic Checklist

LG 1 (COMPLETE). All pertinent information should be included or referenced.
LG 2 (RELEVANT). All information must be relevant to the software product.
LG 3 (BRIEF). Information must be stated succinctly.
LG 4 (CLEAR). Information must be clear to all reviewers and users of the document.
LG 5 (CORRECT). Information does not contain errors.
LG 6 (COHERENT). Information must be consistent with all other information in the document and its source document(s).
LG 7 (UNIQUE). Ideas must be described once and referenced afterward.

Adapted from Gilb and Graham (1993) [GIL 93]

In the third step of the desk-check process, the reviewers verify the document and record their comments on the review form.

The author reviews the comments as part of step 4.

If the author agrees with all the comments, he incorporates them into his document. After this meeting, one of three options should be considered: the comment is incorporated as is, the comment is ignored, or it is incorporated with modifications.

Next step, the author can make the corrections and note

the effort spent reviewing and correcting the document, that is, the time spent by the reviewers as well as the time spent by the author to correct the document and conduct the meeting if this is the case.

In the final step, the author completes the review form illustrated in Figure 5.9.

ENTRY CRITERIA
- The document is ready for a review

INPUT
- Software product to review

DC 100. Plan the Desk-Check
Author:
- Identifies reviewers
- Chooses the checklist(s) to use
- Completes the first part of the review form

DC 110. Send documents to reviewers
Author:
- Provides the following documents to the reviewers:
    - Software product to review
    - Review form
    - Checklist(s)

DC 120. Review the software product
Reviewers:
- Check the software product against the checklist
- Complete the review form with
    - comments
    - effort to conduct the review
- Sign and return the form to the author

DC 130. Call a meeting (if needed)
Author:
- Reviews the comments
    - If the author agrees with all the comments, they are incorporated in the software product
    - If the author does not agree with all the comments, or believes some comments have a significant impact, then the author:
        - Convenes a meeting with the reviewers
        - Leads the meeting to discuss the comments and determine course of action:
            - Incorporate the comment as is
            - Ignore the comment
            - Incorporate the comment with modifications

DC 140. Correct the software product
The author incorporates the comments received.

DC 150. Complete the review form
Author:
- Completes the review form with:
    - Total effort (i.e., by all the reviewers) required to review the software product
    - Total effort required to correct the software product
- Signs the review form

EXIT CRITERIA
- Corrected software product

OUTPUT
- Corrected software product
- Completed and signed review form

MEASURE
- Effort required to review and correct the software product (person hours).

Desk check review form

Name of author: _____ Review date (yyyy-mm-dd): _____
Document title: _____ Reviewer name: _____
Document version: _____

Comment No.	Document page	Line # / location	Comments	Disposition of comments*	Remarks
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					

Disposition of comment: Inc: Incorporate as is; NOT: Not incorporate, MOD: Incorporate with modification

Effort to review document (hour): _____
Effort to correct document (hour): _____

Signature of reviewer: _____

Signature of author: _____

Figure 5.9 Desk-Check review form.

Topic 6.8 The IEEE 1028 Standard

The IEEE 1028-2008 Standard for Software Reviews and Audits [IEE 08b] describes

five types of reviews and audits and the procedures required for the completion of each type of review and audit.

The purpose of this standard is to define reviews and systematic audits for the acquisition, supply, development, operation and maintenance of software.

This standard describes not only "what to do" but also how to perform a review.

This standard provides descriptions of the particular types of reviews and audits included in the standard as well as tips.

a) Introduction to review: describes the objectives of the systematic review and provides an overview of the systematic review procedures;

b) Responsibilities: defines the roles and responsibilities needed for the systematic review.

c) Input: describes the requirements for input needed by the systematic review;

d) Entry criteria: describes the criteria to be met before the systematic review can begin, including the following:
   1) Authorization,
   2) Initiating event;

e) Procedures: details the procedures for the systematic review, including the following:
   1) Planning the review;
   2) Overview of procedures;
   3) Preparation;
   4) Examination/evaluation/recording of results;
   5) Rework/follow-up;

f) Exit criteria: describe the criteria to be met before the systematic review can be considered complete;

g) Output: describes the minimum set of deliverables to be produced by the systematic review.

Topic 6.9 IEEE 1028 -The types of reviews and audits

- management review: a systematic evaluation of a software product or process performed by or on behalf of the management that monitors progress, determines the status of plans and schedules, confirms requirements and their system allocation, or evaluates the effectiveness of the management approaches used to achieve fitness for purpose;
- technical review: a systematic evaluation of a software product by a team of qualified personnel that examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards;
- inspection: a visual examination of a software product to detect and identify software anomalies including errors and deviations from standards and specifications;
- walk-through: a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about any anomalies, violation of development standards, and other problems;
- audit: an independent assessment, by a third party, of a software product, a process or a set of software processes to determine compliance with the specifications, standards, contractual agreements, or other criteria.

Topic 6.10 WALK-THROUGH

"The purpose of a walk-through is to evaluate a software product.

A walk-through can also be performed to create discussion for a software product"

Objectives of the walk-through:

- – find anomalies;
- – improve the software product;
- – consider alternative implementations;
- – evaluate conformance to standards and specifications;
- – evaluate the usability and accessibility of the software product.

Topic 6.11 Usefulness of a Walk-Through

- – identify errors to reduce their impact and the cost of correction;
- – improve the development process;
- – improve the quality of the software product;
- – reduce development costs;
- – reduce maintenance costs.

Table 5.2 Characteristics of Reviews and Audits Described in the IEEE 1028 Standard

	Management review	Technical review	Inspection	Walk-through	Audit
Objective	Monitor progress	Evaluate conformance to specifications and plans	Find anomalies; verify resolution; verify product quality	Find anomalies, examine alternatives; improve product; forum for learning	Independently evaluate conformance with objective standards and regulations
Recommended group size	Two or more people	Two or more people	3-6	2–7	1–5
Volume of material	Moderate to High	Moderate to High	Relatively low	Relatively low	Moderate to High
Leadership	Usually the responsible manager	Usually the lead engineer	Trained facilitator	Facilitator or author	Lead auditor
Management participates	Yes	When management evidence or resolution may be required	No	No	No; however management may be called upon to provide evidence
Output	Management review documentation	Technical review documentation	Anomaly list, anomaly summary, inspection documentation	Anomaly list, action items, decisions, follow-up proposals	Formal audit report; observations, findings, deficiencies

Source: Adapted from IEEE 1028 [IEE 08b].

Topic 6.12 INSPECTION REVIEW

This section briefly describes the inspection process that Michael Fagan developed at IBM in the 1970s to increase the quality and productivity of software development.

The purpose of the inspection, according to the IEEE 1028 standard, is to detect and identify anomalies of a software product including errors and deviations from standards and specifications.

Throughout the development or maintenance process, developers prepare written materials that unfortunately have errors. It is more economical and efficient to detect and correct errors as soon as possible. Inspection is a very effective method to detect these errors or anomalies.

Topic 6.13 According to the IEEE 1028 standard, inspection allows us to

a) verify that the software product satisfies its specifications;

b) check that the software product exhibits the specified quality attributes;

c) verify that the software product conforms to applicable regulations, standards, guidelines, plans, specifications, and procedures;

d) identify deviations from provisions of items (a), (b), and (c);

e) collect data, for example, the details of each anomaly and effort associated with their identification and correction;

f) request or grant waivers for violation of standards where the adjudication of the type and extent of violations are assigned to the inspection jurisdiction;

g) use the data as input to project management decisions as appropriate (e.g., to make trade-offs between additional inspections versus additional testing).

Topic 6.14 Major steps of the inspection process, Each step is composed of a series of inputs, tasks and outputs.

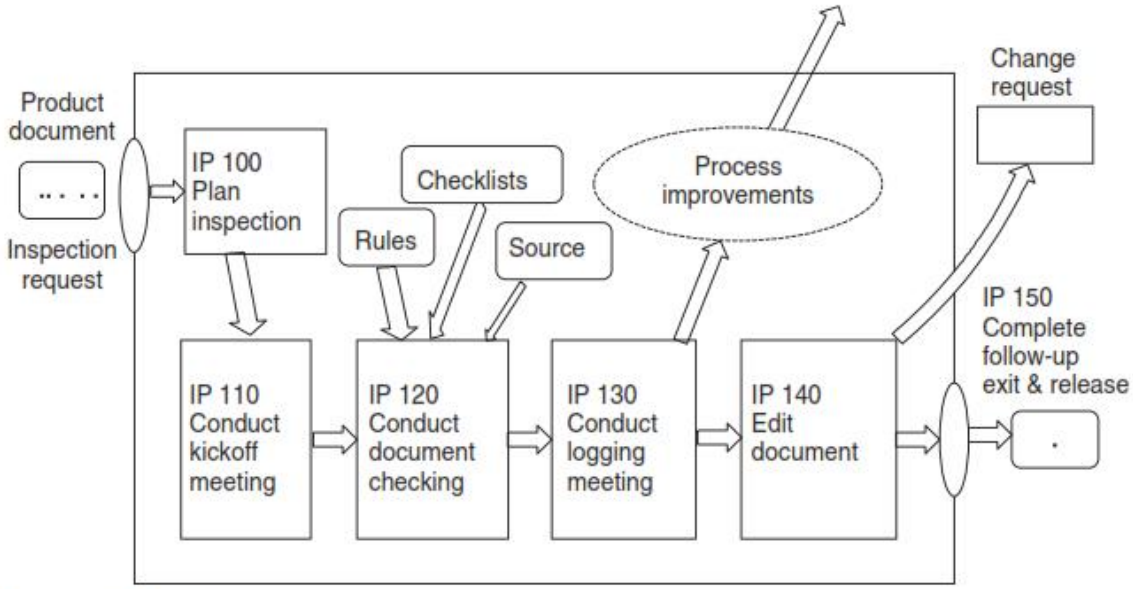


Figure 5.12 The inspection process.

Topic 6.15 PROJECT LAUNCH REVIEWS AND PROJECT

Topic 6.16 ASSESSMENTS

In the SQAP of their projects, many organizations plan a project launch or kick-off meeting as well as a project assessment review, also called a lessons learned review.

Topic 6.17 Project Launch Review

The project launch review is a management review of: the milestone dates, requirements, schedule, budget constraints, deliverables, members of the development team, suppliers, etc.

Some organizations also conduct kick-off reviews at the beginning of each of the major phases of the project when projects are spread over a long period of time

Before the start of a project, team members ask themselves the following questions:

who will the members of my team be? Who will be the team leader? What will my role and responsibilities be? What are the roles of the other team members and their responsibilities? Do the members of my team have all the skills and knowledge to work on this project?

Topic 6.18 MEASURES

Measures are mainly used to answer the following questions:

– How many reviews were conducted?

– What software products have been reviewed?

– How effective were the reviews (e.g., number of errors detected by number of hours for the review)?

– How efficient were the reviews (e.g., number of hours per review)?

– What is the density of errors in software products?

– How much effort is devoted to reviews?

– What are the benefits of reviews?

The measures that allow us to answer these questions are:

– number of reviews held;

– identification of the revised software product;

– size of the software product (e.g., number of lines of code, number of pages);

– number of errors recorded at each stage of the development process;

– effort assigned to review and correct the defects detected.

Topic 6.19 SELECTING THE TYPE OF REVIEW

To determine the type of review and its frequency, the criteria to be considered are:

The risk associated with the software to be developed, the criticality of the software, software complexity, the size and experience of the team, the deadline for completion, and software size.

Table 5.5 Example of a Matrix for the Selection of a Type of Review

	Technical drivers—complexity		
Product	Low	Medium	High
Software requirements	Walk-through	Inspection	Inspection
Design	Desk-check	Walk-through	Inspection
Software code and unit test	Desk-check	Walk-through	Inspection
Qualification test	Desk-check	Walk-through	Inspection
User/operator manuals	Desk-check	Desk-check	Walk-through
Support manuals	Desk-check	Desk-check	Walk-through
Software documents, for example, Version Description Document (VDD), Software Product Specification (SPS), Software Version Description (SVD)	Desk-check	Desk-check	Desk-check
Planning documents	Walk-through	Walk-through	Inspection
Process documents	Desk-check	Walk-through	Inspection

Tools for Conducting Reviews

Several free software tools are available to facilitate code review:

- Idutils: an indexing tool that allows the creation of a database of identifiers used in a program;
- Egrep: a tool to search regular expression patterns in text files;
- Find: allows a system file to be viewed;
- Diff: to compare two files and show differences;
- Cscope: a C-code browser;
- LXR: a web interface to explore the source code online and offer cross-reference.

Topic 6.20 AUDITS

- One of the most formal types of reviews.
- Different types of conformity certificates (e.g., audits) respond to different needs, such as the needs of an organization that develops software products or those of a client of a software product supplier.
- The independence level of the auditor as well as the cost varies depending on the audit type.
- **Management System**

System to establish policy and objectives and to achieve those objectives.

- **Audit Criteria**

Set of policies, procedures or requirements used as references against which objective evidence is compared.

- **Audit Evidence**

Records, statement of fact or other information, which are relevant to the audit criteria and verifiable.

- **Audit**

Systematic, independent and documented process for obtaining audit evidence and evaluating it objectively to determine the extent to which the audit criteria are fulfilled.

Topic 6.21 Internal audits

- Called first party audits
- Conducted by the organization itself, or on its behalf, for management review and other internal purposes (e.g. to confirm the effectiveness of the management system or to obtain information for the improvement of the management system).
- Internal audits can form the basis for an organization's self-declaration of conformity.

Topic 6.22 External audits

- Include second and third party audits. Second party audits are conducted by parties having an interest in the organization, such as customers, or by other persons on their behalf.
- Third party audits are conducted by independent auditing organizations, such as regulators or those providing certification.

There are different types of audit:

– audits to verify the compliance to a standard, such as the audits described in International Organization for Standardization (ISO) standards such as ISO

9001 and IEEE 1028;

– compliance audits for a model such as the Capability Maturity Model Integration (CMMI) that are used to select a supplier before awarding a contract or assess a supplier during a contract;

– audits ordered by the management team of the organization to verify the progress of a project against its approved plan.

Topic 6.23 Why audit?

Software project audits are usually requested by management to ensure that the software team and contracted suppliers:

- – know their duties and obligations toward the public, their employers, their customers, and their colleagues;
- – use the processes, practices, techniques, and normalized methods suggested by the company;
- – reveal any deficiencies and shortcomings in daily operations and try to identify required corrective actions (CAs);
- – are encouraged to develop a personal training plan for their professional skills;
- – are monitored in the course of their work on high profile projects of the company.

Topic 6.24 TYPES OF AUDITS

Topic 6.25 Internal Audit

- first-party audit, can be useful for a software supplier wanting to obtain an ISO 9001 certification.
- It is the least expensive approach to prepare for conformity to an international standard.

Topic 6.26 Second-Party Audit

conducted by parties having an interest in the organization, such as customers, or by other persons on their behalf.

Third party audits are conducted by independent auditing organizations, such as regulators or those providing certification.

To emphasize that ISO does not offer certification services per say. It does not deliver certificates either.

It is the ISO 19011 [ISO 11g] standard entitled “Guidelines for auditing management systems” and the ISO/IEC 17021-1 [ISO 15a] standard, entitled “Conformity assessment Requirements for bodies providing audit and certification of management systems Part 1: Requirements” that are used to assess the compliance to a standard such as ISO 9001.

Topic 6.27 Project Assessment and Control Process

- To assess if the plans are aligned and feasible; determine the status of the project, technical and process performance; and direct execution to help ensure that the performance is according to plans and schedules, within projected budgets, to satisfy technical objectives.
- One important mandatory task of this process is the conduct of management and technical reviews, audits, and inspections.

Topic 6.28 CORRECTIVE ACTIONS

- After an internal or external audit, an organization must perform CAs to correct the observed deficiencies.
- It is also possible to treat the preventive actions, an incident report, and customer complaints using the CA process.
- A CA aims at eliminating potential causes of non- conformity, a defect, or any other adverse event to prevent their recurrence.
- This process should cover software products, agreements and software development plans.

Corrective Action

Action to eliminate the cause of a nonconformity and to prevent recurrence.

Note 1: There can be more than one cause for a nonconformity.

Note 2: Corrective action is taken to prevent recurrence whereas preventive action is taken to prevent occurrence.

ISO 9000 [ISO 15b]

An intentional activity that realigns the performance of the project work with the project management plan.

PMBOK® Guide [PMI 13]

Preventive Action

An intentional activity that ensures the future performance of the project work is aligned with the project management plan.

PMBOK® Guide [PMI 13]

Topic 6.29 Corrective Actions Process

- The problems encountered when developing systems that include software, or that occur during their operation, can come from defects in the software, in the development process itself, or in the hardware of the system.
- To facilitate the identification of problem sources and apply appropriate CAs, it is desirable that a centralized system is developed to track issues through to resolution and to determine their root cause.

Topic 6.30 A CA process, in a closed loop, may include the following elements:

– inputs: for example, an audit report, a non-compliance, or a problem report;

– activities:

◦ register non-conformities in the issue tracking tool used by the organization,

◦ analyze and validate the problem to ensure that the organization's resources are not wasted,

◦ classify and prioritize the issue/problem,

◦ analyze the problems to conduct trend analysis that can be identified and addressed,

◦ propose a solution to the problem,

◦ solve the problem and ensure that the resolution does not cause other problems, or if the non-compliance issue cannot be resolved within the project,refer it to the appropriate management level,

◦ verify the problem resolution,

◦ inform stakeholders of the problem resolution,

◦ archive the problem documentation,

◦ update the information in the issue tracking tool;

– outputs: for example, the resolution file, a corrected version of the software.

Problem report

Priority:_____ Project name: _____ Date: _____

Process name: _____ Phase number: ___Raised by: _____

Number of days to answer: _____ Close date: _____

Number of days to fix this problem: _____

Finding: _____

Requirement/Standard impacted:
_____

Immediate solution proposed:
_____

Root cause: _____

Permanent solution proposed:
_____

Acceptance date of permanent solution: _____

Follow-up action (if necessary):
_____

Figure 6.5 Problem report and resolution proposal form.

7. CS 11 & 12 Software Quality Assurance and Testing Module 7

Topic 7.1 Effective Test Management and Planning

- During the testing phase of software development, testing activities are managed well to complete the testing process smoothly and on time as well.
- **Test Management** is a process where testing activities are managed to ensure high-quality and high-end testing of software applications.
- This method consists of tracking, organization, controlling processes and checking the visibility of the testing process to deliver a high-quality software application.
- Test management is concerned with both test resource and test environment management.
- It is the role of test management to ensure that new or modified service products meet the business requirements for which they have been developed or enhanced.
- It makes sure the software testing process runs as expected.

Topic 7.2 key elements of test management

- **Test organization**
- **Test planning**
- **Detailed test design and test specifications**
- **Test monitoring and assessment**
- **Product quality assurance**

Topic 7.3 TEST ORGANIZATION

- It is the process of setting up and managing a suitable test organizational structure and defining explicit roles.
- The project framework under which the testing activities will be carried out is reviewed, high-level test phase plans are prepared, and resource schedules are considered.
- Test organization also involves the determination of configuration standards and defining the test environment.
- Since testing is viewed as a process, it must have an organization such that a testing group works for better testing and high quality software.
- The testing group is responsible for the following activities:
    - ·**Maintenance and application of test policies „**
    - · **Development and application of testing standards „**
    - · **Participation in requirement, design, and code reviews „**
    - · **Test planning „**
    - · **Test execution „**

- · **Test measurement „**
- · **Test monitoring „**
- · **Defect tracking „**
- · **Acquisition of testing tools „**
- · **Test reporting**

The staff members of such a testing group are called test specialists or test engineers or simply testers.

A tester is not a developer or an analyst. He does not debug the code or repair it.

He is responsible for ensuring that testing is effective and quality issues are being addressed.

Topic 7.4 Skills a Tester

1. Personal and Managerial Skills: -

- Testers must be able to contribute in policy-making and planning the testing activities.
- Testers must be able to work in a team.
- Testers must be able to organize and monitor information, tasks, and people.
- Testers must be able to interact with other engineering professionals, software quality assurance staff, and clients.
- Testers should be capable of training and mentoring new testers.
- Testers should be creative, imaginative, and experiment-oriented.
- Testers must have written and oral communication skills.

2. Technical Skills: -

- Testers must be technically sound, capable of understanding software engineering principles and practices.
- Testers must be good in programming skills.
- Testers must have an understanding of testing basics, principles, and practices.
- Testers must have a good understanding and practice of testing strategies and methods to develop test cases.
- Testers must have the ability to plan, design, and execute test cases with the goal of logic coverage.
- Testers must have technical knowledge of networks, databases, operating systems, etc. needed to work in a the project environment.
- Testers must have the knowledge of configuration management.
- Testers must have the knowledge of test ware and the role of each document in the testing process.
- Testers must have known about quality issues and standards.

Topic 7.5 STRUCTURE OF TESTING GROUP

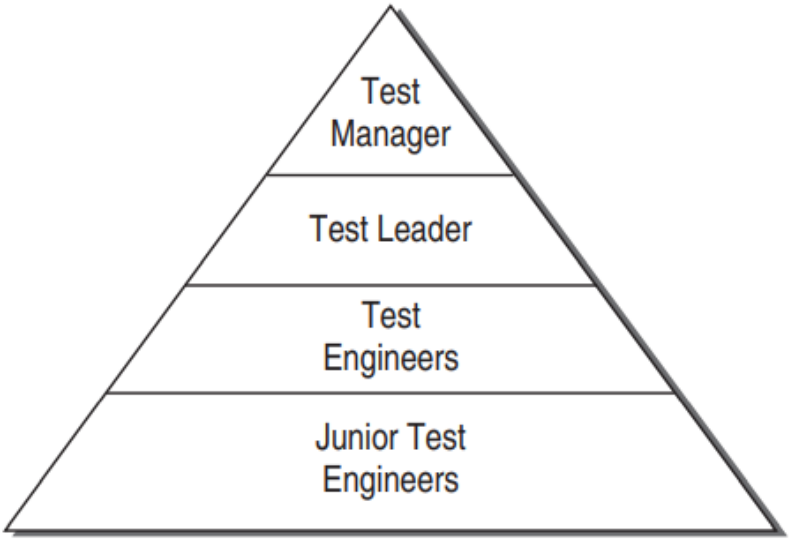


Figure 9.1 Testing Group Hierarchy

Test Manager: A test manager occupies the top level in the hierarchy.

- i. He is the key person in the testing group who will interact with project management, quality assurance, and marketing staff.
- ii. Takes responsibility for making test strategies with detailed master planning and schedule.
- iii. Interacts with customers regarding quality issues.
- v. Acquires all the testing resources including tools.
- v. Monitors the progress of testing and controls the events.
- vi. Participates in all static verification meetings.
- vii. Hires, fires, and evaluates the test team members.

Test Leader: Assists the test manager in meeting testing

and quality goals. To lead a team of test engineers who

are working at the leaf-level of the hierarchy.

- i. Planning the testing tasks given by the test manager.
- ii. Assigning testing tasks to test engineers who are working under him.
- iii. Supervising test engineers.
- iv. Helping the test engineers in test case design, execution, and reporting.
- v. Providing tool training, if required.
- vi. Interacting with customers.

Test Engineers: Test engineers are highly experienced testers.

- i. Designing test cases.
- ii. Developing test harness.
- iii. Set-up test laboratories and environment.
- iv. Maintain the test and defect repositories.

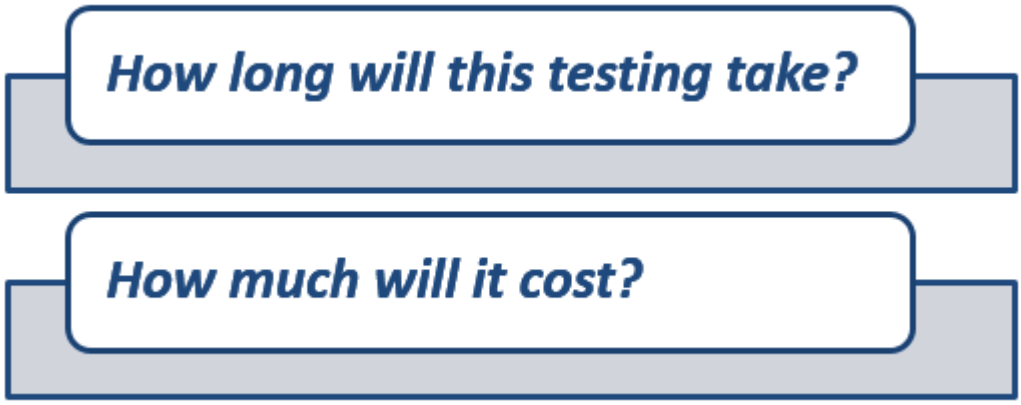
Junior Test Engineers: -

- Junior test engineers are newly hired testers. They usually are trained about the test strategy, test process, and testing tools.
- They participate in test design and execution with experienced test engineers.

Topic 7.6 Test Estimation and Scheduling

- Test estimation techniques refer to the methods and approaches used to determine or estimate the effort, time, and resources required for testing activities in software development projects.
- Software test estimation is a managerial task that involves assessing and approximating the required time, resources, and costs for executing tests in a specific environment.
- It serves as a projection that aids in preventing time constraints and exceeding budgets.

Topic 7.7 Why Test Estimation?



Project Planning: With correct estimations done, the overall project timeline can be kept under check. Project managers can create realistic schedules and allocate resources as per need if they know the time required for testing activities well in advance. This allows for effective coordination with development and other project activities.

Resource Allocation: With test estimations in place, the resource allocations: namely number of testers, testing tools, and testing environments required, can be allocated diligently and efficiently. It helps to ensure that overallocation or underutilization of resources is avoided.

Budgeting and Cost Control: Budgets are crucial to any project's success. Expenses like personnel costs, infrastructure costs, and tooling costs can be estimated well if testing efforts are estimated

efficiently. By having a clear understanding of the expected costs, organizations can create accurate budgets, monitor expenses, and prevent cost overruns.

Risk Management: Estimation can also help build visibility into any potential risks associated with testing including areas that may require additional attention or resources. This allows for risk mitigation strategies to be implemented proactively.

Stakeholder Expectations: Estimation helps set realistic expectations with project stakeholders, including clients, managers, and development teams.

Project Optimization: Accurate estimation allows for better planning and optimization of testing activities. It helps identify opportunities for process improvements, resource optimization, and automation, leading to increased efficiency and productivity in the testing process.

Topic 7.8 What to Estimate?

Resources: Resources are required to **carry out** any project tasks. They can be people, equipment, facilities, funding, or anything else capable of definition required for the completion of a project activity.

Times : Time is the most valuable resource in a project. Every project has a deadline to delivery.

Human Skills : Human skills mean the **knowledge** and the **experience** of the Team members. They affect to your estimation. For example, a team, whose members have low testing skills, will take more time to finish the project than the one which has high testing skills.

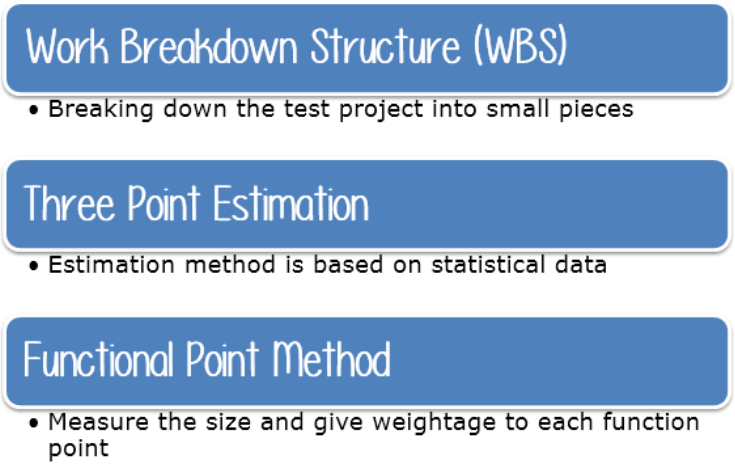
Cost: Cost is the project **budget**. Generally speaking, it means **how much money** it takes to finish the project.

Topic 7.9 How to estimate?

Topic 7.10 Software Test Estimation Techniques

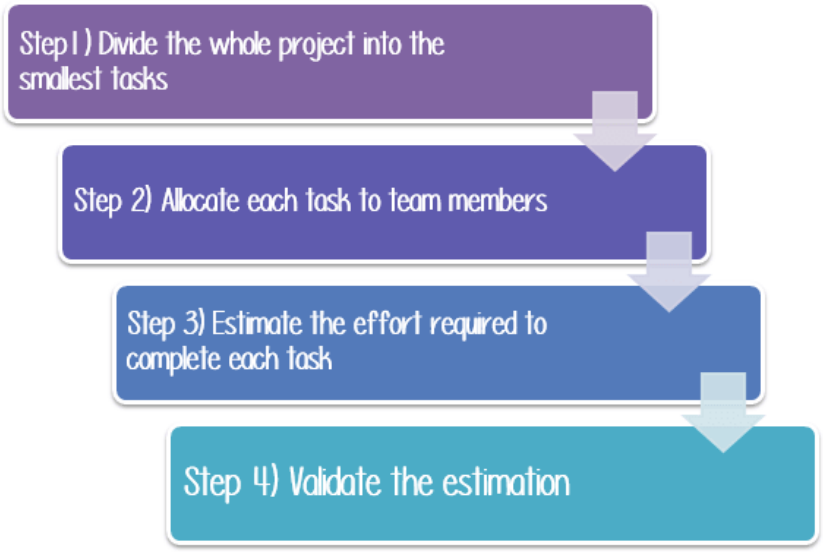
Work Breakdown Structure	It is a method of breaking down large tasks into smaller, easily executable groups
3-Point Software Estimation Test	Tasks are broken down into smaller tasks, and then each task is estimated based on 3 points – Best case, Most likely and Worst case estimates
Wideband Delphi Method	It is a technique where an expert set of panel comes together to determine the most probable outcome and they have a common consensus
Functional Point Analysis	Large tasks are broken down into small tasks. Each small task is then estimated based on size, cost and duration of project
Agile Estimation	In these techniques, the current data and past experience are used for estimation and new information is continuously integrated into the project to refine the process of estimation
Distribution in Percentage	Here, every stage is assigned or assessed in terms of percentages. This is to find how much effort should be put into each stage of the testing cycle.

Topic 7.11 Example : Bank Case Study



- **Combine these techniques to find the estimate.**

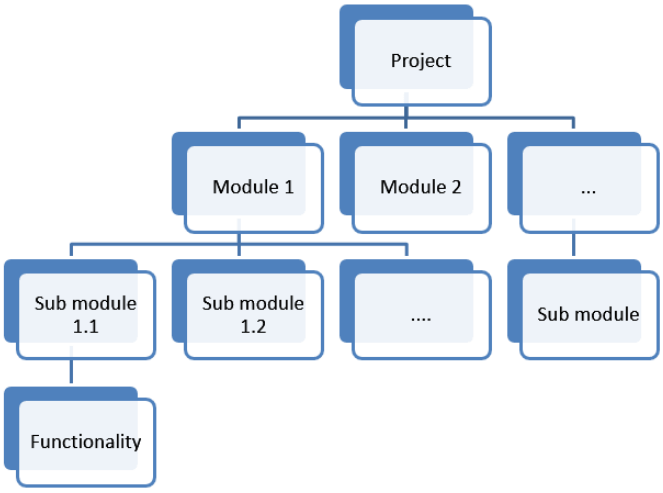
Following 4 Step process to arrive at an estimate



Topic 7.12 Step 1) Divide the whole project task into subtasks

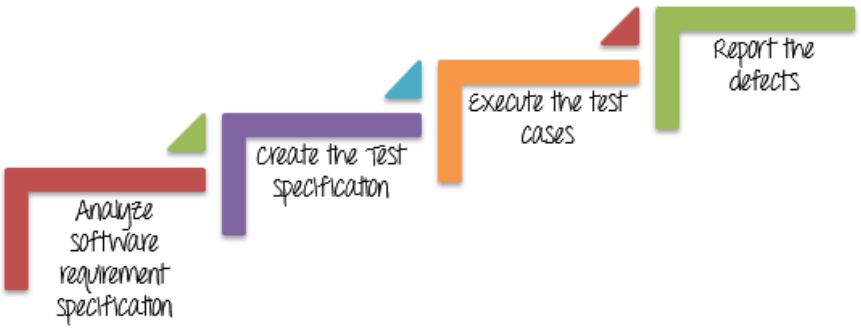
Work Breakdown Structure technique. : a complex project is divided into modules. The modules are divided into sub-modules. Each sub-module is further divided into functionality. It means divide the whole project task into the **smallest** tasks.

- Using Work Break Down structure to break out the Bank project into 5 smaller tasks



- After that, you can break out each task to the **subtask.**

The purpose of this activity is create tasks **detailed** as **possible**.



Task	Sub task
Analyze software requirement specification	Investigate the soft requirement specs
	Interview with the developer & other stakeholders to know more about the website
Create the Test Specification	Design test scenarios
	Create test cases
	Review and revise test cases
Execute the test cases	Build up the test environment
	Execute the test cases
	Review test execution results
Report the defects	Create the Defect reports
	Report the defects

Topic 7.13 Step 2) Allocate each task to team member

Each task is assigned to the **appropriate** member in the project team.

Task	Members
Analyze software requirement specification	All the members
Create the test specification	Tester/Test Analyst
Build up the test environment	Test Administrator
Execute the test cases	Tester, Test Administrator
Report defects	Tester

Topic 7.14 Step 3) Effort Estimation For Tasks

There are 2 techniques which you can apply to estimate the effort for tasks

- **Functional Point Method**

- **Three Point Estimation**

Topic 7.15 Method 1) **Function** Point Method **to estimate the effort for tasks**

- The Test Manager estimates Size, Duration, and Cost for the tasks

Topic 7.16 Step A) Estimate size for the task :

In Step 1, you already have broken the whole project task into small task by using WBS method. Now you estimate the size of those tasks.

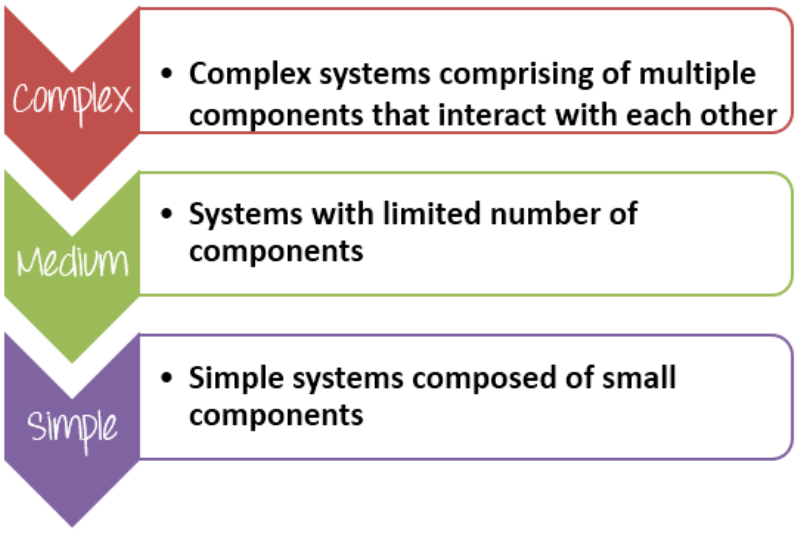
Example task: "**Create the test specification**"

The size of this task depends on the functional size of the system under test.

The functional size reflects the **amount** of functionality that is relevant to the user.

The more **number** of functionality, the more **complex** system is.

Prior to start actual estimating tasks effort, functional points are divided into three groups like **Complex**, **Medium Simple** as following:



Based on the complex of software functions, the Test Manger has to give enough **weightage** to each functional point. For example

Group	Weightage
Complex	5
Medium	3
Simple	1

Let's take a simple example exercise

- By looking at SRS the software specification of Bank website, where the software engineer have already described the software modules in detail, we can determine the **complexity** of website's features by giving the weightage for each modules.
- More complex the function point, more is the effort to test it is. The website is divided into **12 function** points, you can determine the **complexity** of each function points as follows-

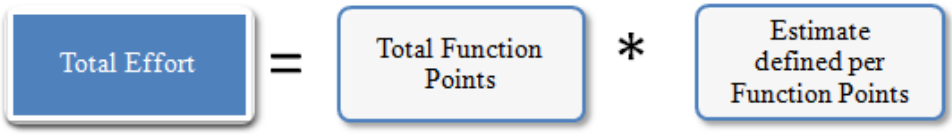
No.	Module Name	Applicable Roles	Description	Weightage
1.	Balance Enquiry	Manager Customer	**Customer:** A customer can have multiple bank accounts. He can view balance of his accounts only **Manager:** A manager can view balance of all the customers who come under his supervision	3
2.	Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his "own" account to any destination account. **Manager:** A manager can transfer funds from any source bank account to destination account	5
3.	Mini Statement	Manager Customer	A Mini statement will show last 5 transactions of an account **Customer:** A customer can see mini-statement of only his "own" accounts **Manager:** A manager can see mini-statement of any account	3
4.	Customized Statement	Manager Customer	A customized statement allows you to filter and display transactions in an account based on date, transaction value **Customer:** A customer can see Customized-statement of only his "own" accounts **Manager:** A manager can see Customized - statement of any account	5

5.	Change Password	Manager Customer	**Customer:** A customer can change password of only his account. **Manager:** A manager can change password of only his account. He cannot change passwords of his customers	1
6.	New Customer	Manager	**Manager:** A manager can add a new customer. **Manager:** A manager can edit details like address, email, telephone of a customer.	3
7.	New Account	Manager	Currently system provides 2 types of accounts • Saving • Current A customer can have multiple saving accounts (one in his name, other in a joint name etc). He can have multiple current accounts for different companies he owns.	5
8.	Edit Account	Manager	**Manager:** A manager can add an edit account details for an existing account	1
9.	Delete Account	Manager	**Manager:** A manager can add a delete an account for a customer.	1
10.	Delete Customer	Manager	A customer can be deleted only if he/she has no active current or saving accounts **Manager:** A manager can delete a customer.	1
11.	Deposit	Manager	**Manager:** A manager can deposit money into any account. Usually done when cash is deposited at a bank branch.	3
12.	Withdrawal	Manager	**Manager:** A manager can withdraw money from any account. Usually done when cash is withdrawn at a bank branch.	3

Topic 7.17 STEP B) Estimate duration for the task

After classifying the **complexity** of the function points, you have to estimate the **duration** to test them. Duration means **how much** time needs to finish the task.

- **Total Effort**: The effort to completely test all the functions of the website
- **Total Function Points**: Total modules of the website
- **Estimate defined per Function Points**: The average effort to complete one function points. This value depends on the **productivity** of the member who will take in charge this task.



Suppose your project team has estimated defined per Function Points of 5 hours/points. You can estimate the total effort to test all the features of Bank website as follows:

	Weightage	# of Function Points	Total
Complex	5	3	15
Medium	3	5	15
Simple	1	4	4
Function Total Points			**34**
Estimate define per point			**5**
Total Estimated Effort (Person Hours)			**170**

So the total effort to complete the task **"Create the test specification"** of Bank website is is around 170 man-hours.

1 man hour = work completed in an hour of uninterrupted effort by an average worker.

- Once you understand the effort that is required, you can assign resources to determine how long the task will take (duration), and then you can estimate labor and non-labor costs.
- Above example also shows the importance of the member in your team.
- If you have **talented** and **experienced** members, you can finish the assigned task in the **small** time, and your project will finish at the deadline or sooner.

Topic 7.18 STEP C) Estimate the cost for the tasks

"**How much does it cost?"**

- Suppose, on average your team salary is $5 per hour.
- The time required for "Create Test Specs" task is 170 hours. Accordingly, the cost for the task is 5*170= $850.
- Now you can calculate budget for other activities in WBS and arrive at overall budget for the project.

Topic 7.19 Method 2) Three Point Estimation

- Three-Point estimation is one of the techniques that could be used to estimate a task.
- The simplicity of the Three-point estimation makes it a very useful tool for a Project Manager that who wants to estimate.
- In three-point estimation, **three** values are produced initially for every task based on **prior experience** or **best-guesses** as follows

When estimating a task, the Test Manager needs to provide three values, as specified above.

The three values identified, estimate what happens in an **optimal state**, what is the **most likely**, or what we think it would be the **worst case scenario.**

Example: "**Create the test specification**", for bank website

You can estimate as following

The **best case** to complete this task is **120** man-hours (around 15 days). In this case, you have a talented team, they can finish the task in smallest time.

The **most likely** case to complete this task is **170** man-hours (around 21 days). This is a normal case, you have enough resource and ability to complete the task

The **worst case** to complete this task is **200** man-hours (around 25 days). You need to perform much more work because your team members are not experienced.

Now, assign the value to each parameter as below

$$a = 120 \quad m = 170 \quad b = 200$$

The effort to complete the task can be calculated using **double-triangular distribution** formula as follows-

- Parameter E is known as **Weighted Average.**
- It is the estimation of the task "Create the test specification".

$$E = (a + 4m + b)/6$$

$$E = (120 + 4 * 170 + 200)/6$$

$$E = 166.6\ (man - hours)$$

In the above estimation, you just determine a **possible** and not a **certain** value, we must know about the **probability** that the estimation is correct. You can use the other formula:

SD mean Standard Deviation, this value could give you the information about the **probability** that the estimation is correct.

$$SD = (b - a)/6$$

$$SD = (200 - 120)/6$$

$$SD = 13.33\ (man - hours)$$

- Now you can conclude the estimation for the task "Create the test specification"
- To complete the task "Create the test specification" of Bank website, you need **166.6 ± 13.33** Man-hour (153.33 to 179.99 man-hour)

Topic 7.20 Step 4) Validate the estimation

- Once you create an aggregate estimate for all the tasks mentioned in the WBS, you need to forward it to the **management board**, who will **review** and **approve** it.
- The member of management board could comprise of the CEO, Project Manager & other stakeholders.
- The management board will review and discuss your estimation plan with you. You may explain them your estimation **logically** and **reasonably** so that they can approve your estimation plan.

Topic 7.21 Test estimation best practices

- **Add some buffer time:**
- **Account Resource planning in estimation**
- **Use the past experience as reference**
- **Stick to your estimation**

Topic 7.22 Test Data Management

- The process of planning, creating, and maintaining the datasets used in testing activities, ensuring that they are the right data for the right test case, in the right format, and available at the right time.
- Test data is the set of input values used during the testing process of an application (software, web, mobile application, API, etc).
- These values represent what a user would enter the system in a real-world scenario.
- Testers usually can write a test script to automatically and dynamically identify the right type of values to put into the system and see how it responds to those data.
- Example : Test data for the testing of a login page
  - **Username** column and a **Password** column.
  - A test script or automation testing tool can open the Login page, identify the Username field, the Password field, then input the values
  - Can have hundreds to thousands of such credential pairs representing unique test scenarios.

Username	Password
user_123	Pass123!
testuser@email	Secret@321
admin_user	AdminPass#
jane_doe	JaneDoePass

- You can have hundreds to thousands of such credential pairs representing unique test scenarios.
- But having a huge database does not immediately mean all of it is high-quality.

Topic 7.23 Criteria to evaluate test data quality

- **Relevance:** it makes sense to have test data that accurately reflects the scenario being tested.

  Imagine testing the response of the login page when users enter the wrong set of credentials, but the test data being used is actually the correct, stored-in-database credentials. This returns inaccurate results.

- **Availability:** what's the point of having thousands of relevant data points yet you can't retrieve them for testing activities?

Usually QA teams have clearly defined role-based access for test data, so TDM activities are also about assigning the right level of access to the right personnel.

- **Updated:** software constantly changes, bringing with it new complexities and dependencies.

The responsibility of QA teams is to be aware of those updates and make changes to the test data accordingly to ensure that results accurately reflect the current state of the software.

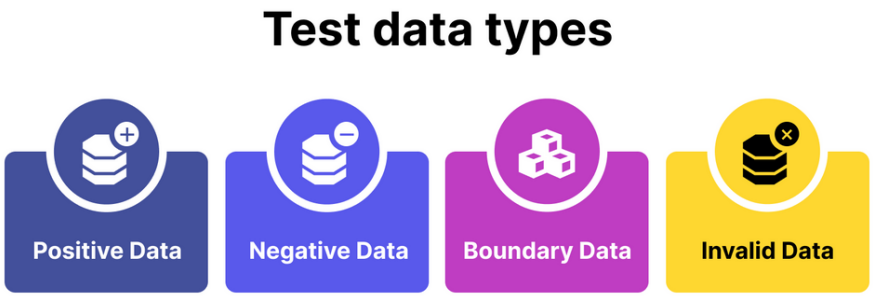
- **Compliance:** aside from the technical aspects, we should never forget compliance requirements.

QA teams sometimes leverage directly production data for testing activities due to its instant availability, but production data is a tricky domain: it may contain confidential information protected by GDPR, HIPAA, PCI, or other data privacy focused policies.

PCI DSS, HIPAA, and GDPR are all security standards that protect sensitive information, but they differ in their scope, purpose, and enforcement.

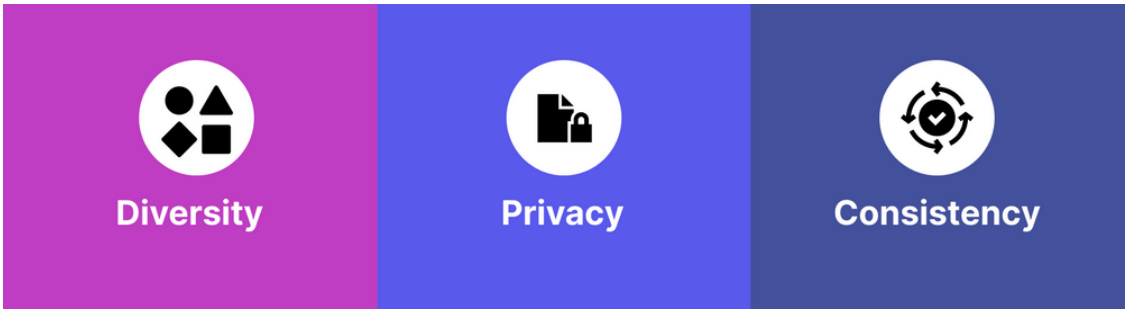
- **Payment Card Industry Data Security Standard** is a set of security standards that protect cardholder data.
- **Health Insurance Portability and Accountability Act** is a federal law that protects patient health information (PHI)
- **General Data Protection Regulation** is a law that protects the personal data of EU citizens.

Topic 7.24 Types of Test Data



- **Positive Test Data:** this type of data consists of input values that are valid and within the expected range and is designed to test how the system behaves under expected and normal conditions. Examples: a set of valid username and password that allows users to login to their account page on an eCommerce site.
- **Negative Test Data:** in contrast with positive data, negative test data consists of input values that are invalid, unexpected, or outside the specified range. It is designed to test how the system behaves when users do something out of the "correct" path intended. Examples: a set of username and password that is too long.
- **Boundary Test Data:** these are values at the edges or boundaries of acceptable input ranges chosen to assess how the system handles inputs at the upper and lower limits of the allowed range.
- **Invalid Test Data:** these are data that does not accurately represent the real-world scenarios or conditions that the software is expected to encounter. It does not conform to the expected format, structure, or rules within a given context.

Topic 7.25 Why Test Data Management?



➢ **Diversity:** high test coverage is synonymous with covering a rich array of test scenarios, and subsequently having test data for all of those scenarios.

A simple registration page, for example, already requires so many datasets to cover all of the possible scenarios that can happen there :

- Valid credentials
- Empty username
- Empty password
- Incorrect username

- SQL injection attempt
- Special characters
- Too long username
- Too long password

- **Data Privacy:** without good TDM practices, testers can risk using PII (personally identifiable information) to test, which is a breach in security.
  - There are so many things you can do in TDM to prevent this from happening, such as data anonymization, which is essentially a process to replace real, sensitive data with similar but fictitious data.
  - If teams decide to use real data, they can mask (i.e. encrypt) specific sensitive data fields, and use only the most necessary.
  - Several teams employ Dynamic Data Masking (DDM) to dynamically mask data fields based on user roles and permissions.
- **Data Consistency:** QA teams also need to ensure that their test data is uniform across the entire systems, adhering to the same format and standards, and even the relationships among the datasets must be continuously maintained over time when the complexity of the system grows.

Topic 7.26 Test Data Management Techniques

1. **Data Masking**
2. **Data Subsetting**
3. **Synthetic Data Generation**

Topic 7.27 Data Masking

- Data masking is the technique used to protect sensitive information in non-production environments by replacing, encrypting, or otherwise “masking” confidential data while retaining the original data's format and functionality.
- Data masking creates a sanitized version of the data for testing and development purposes without exposing sensitive information.

Data Masking Technique	Definition + Examples
Substitution	**Definition**: Replace actual sensitive data with fictional or anonymized values. You can leverage Generative AI for this approach; however, note that creating entirely new data is resource-intensive. **Example**: Replace actual names with randomly generated names (e.g., John Doe).
Shuffling	**Definition**: Randomly shuffle the order of data records to break associations between sensitive information and other data elements. This approach is faster and easier to achieve compared to the Substitution. **Example**: Shuffle the order of employee records, disconnecting salary information from individuals.
Encryption	**Definition**: Use encryption algorithms to transform sensitive data into unreadable ciphertext. Only authorized users with decryption keys can access the original data. This is a highly secured approach to take. **Example**: Encrypt credit card numbers, rendering them unreadable without proper decryption.
Tokenization	**Definition**: Replace sensitive data with randomly generated tokens. Tokens map to the original data, allowing reversible access by authorized users. **Example**: Replace social security numbers with unique tokens (e.g., Token123).
Character Masking	**Definition**: Mask specific characters within sensitive data, revealing only a portion of the information. **Example**: Mask all but the last four digits of a social security number (e.g., XXX-XX-1234).
Dynamic Data Masking	**Definition**: Dynamically control and limit the exposure of confidential data in real-time during query execution. In other words, sensitive data is masked at the moment of retrieval, just before being presented to the user (usually the masking logic is based on user roles). **Example**: Mask salary information in query results for users without financial access rights.

Randomization	**Definition**: Introduce randomness to the values of sensitive data for creating diverse test datasets. **Example**: Randomly adjust salary values within a specified percentage range for a group of employees.

Topic 7.28 Data Subsetting

Data subsetting is a technique to create a smaller yet representative subset of a production database for use in testing and development environments.

Topic 7.29 Data Subsetting : Benefits

- Reduce data volume, especially in organizations with large datasets. For testing purposes, smaller data volume minimizes resource requirements and therefore reduces maintenance needs.
- Preserve data integrity, as subsetting a dataset does not change the relationship between rows, columns, and any entities within it
- Easily include/exclude data based on specific criteria relevant to the team's testing needs, giving them a higher level of control. At the same time, this translates into improved efficiency in terms of data storage, transmission, and processing.

Topic 7.30 Synthetic Data Generation

- Synthetic data generation is the process of creating artificial datasets that simulate real-world data without containing any sensitive or confidential information.
- This approach is usually reserved only for when obtaining real data is challenging (i.e. financial, medical, legal data) or risky data (i.e. employee personal information).
- generating entirely new sets of data for testing purposes is a more practical approach.
- These synthetic datasets aim to simulate the original dataset as closely as possible, and that means capturing its statistical properties, patterns, and relationships.

Topic 7.31 Tools : Test Data Management

- Informatica- Data provisioning, data subsetting, data masking and data profiling are all included.
- Compuware- It intends to make the extraction, masking and delivery of test data simpler.
- Delphix- It can interact with many databases and systems and allows you to build and deliver masked or fake information copies for testing.
- Microfocus Data Express- Sensitive data is hidden and portions of production data are created.
- IBM InfoSphere Optim- It allows to produce, subset and conceal data for testing while preserving the security and privacy of the data.

Topic 7.32 Software Configuration Management

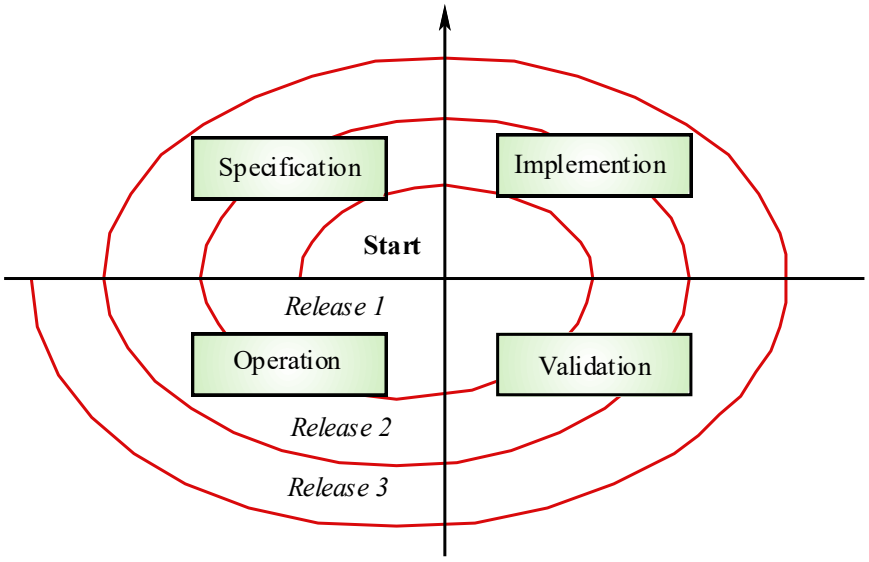
Topic 7.33 Maintenance is Inevitable

- System requirements are likely to change while the system is being developed because their environment is changing
- Systems are tightly coupled to their environment
- When a system is installed it changes the environment and that can change the system requirements
- The delivered system may not meet its requirements
- Systems must be maintained to remain useful in their environment

Topic 7.34 Types of Maintenance

- Corrective Maintenance (21%)
  - making changes to repair defects
- Adaptive Maintenance (25%)
  - making changes to adapt software to external environment changes (hardware, business rules, OS, etc.)
- Perfective Maintenance (50%)
  - extending system beyond its original functional requirements
- Preventative Maintenance (4%)
  - modifying work products so that they are more easily corrected, adapted, or enhanced

Topic 7.35 Spiral Maintenance Model



Topic 7.36 Maintenance Costs

- Usually greater than the development costs (2 to 10 times as much in some cases)
- Affected by both technical and non-technical factors

- Increase as software is maintained and system corruption is introduced
- Aging software can have high support costs (e.g. old languages, compilers, etc.)

Topic 7.37 Maintenance Developer Tasks

- Understand system.
- Locate information in documentation.
- Keep system documentation up to date.
- Extend existing functions.
- Add new functions.
- Find sources of errors.
- Correct system errors.
- Answer operations questions.
- Restructure design and code.
- Delete obsolete design and code.
- Manage changes.

Topic 7.38 Maintenance can be tough

- Limited understanding of hardware and software (maintainer).
- Management priorities (maintenance may be low priority).
- Technical problems.
- Testing difficulties (finding problems).
- Morale problems (maintenance is boring).
- Compromise (decision making problems).

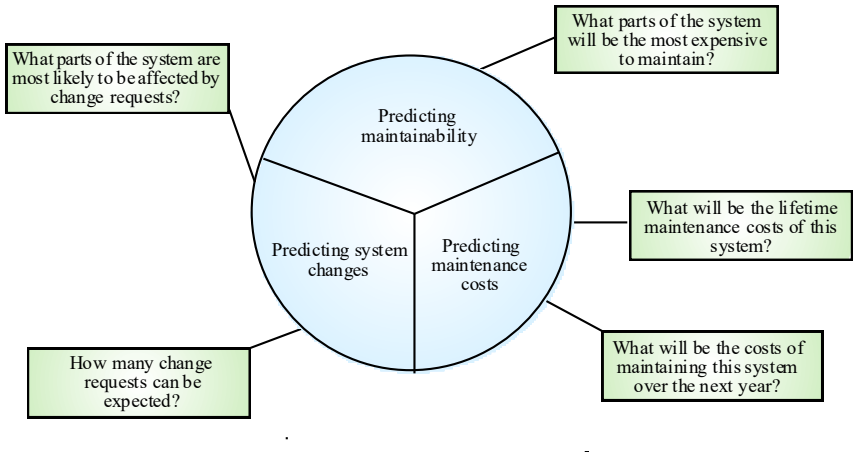
Topic 7.39 Maintenance Cost Factors

- Staff turnover
  - no turnover usually means lower maintenance costs
- Contractual responsibility
  - developers may have no contractual obligation to maintain the delivered system and no incentive to design for future change
- Staff skills
  - maintenance staff are often inexperienced and have limited domain knowledge
- Program age and structure
  - as programs age their structure deteriorates, they become harder to understand and change

Topic 7.40 Maintenance Prediction

- Concerned with determining which parts of the system may cause problems and have high maintenance costs
- Change acceptance depends on the maintainability of the components affected by the change
- Implementing changes degrade system and reduces its maintainability
- Maintenance costs depends on number of changes
- Costs of change depend on maintainability

Topic 7.41 Maintenance Prediction



Topic 7.42 Maintenance Complexity Metrics

- Predictions of maintainability can be made by assessing component complexities
- Most maintenance efforts only affect a small number of system components
- Maintenance complexity depends on
  - complexity of control structures
  - complexity of data structures
  - module size

Topic 7.43 Maintenance Process Metrics

- Maintainability measurements
  - number of requests for corrective maintenance
  - average time required for impact analysis
  - average time to implement a change request
  - number of outstanding change requests
- If any if these increases it may signal a decline in maintainability

Topic 7.44 Maintenance Tools

- Text editors (better than punch cards).

- File comparison tools.
- Compilers and linkage editors.
- Debugging tools.
- Cross reference generators.
- Complexity calculators.
- Control Libraries.
- Full life cycle CASE tools.

Topic 7.45 Configuration Management

- Software changes are inevitable
- One goal of software engineering is to improve how easy it is to change software
- Configuration management is all about change control.
- Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project.
- To ensure quality is maintained the change process must be audited.

Topic 7.46 Software Configuration Items

- Computer programs
  - source
  - executable
- Documentation
  - technical
  - user
- Data
  - contained within the program
  - external data (e.g. files and databases)

Topic 7.47 Baselines

- A work product becomes a baseline only after it is reviewed and approved.
- A baseline is a milestone in software development marked by the delivery of one or more configuration items.
- Once a baseline is established each change request must be evaluated and verified before it is processed.

Topic 7.48 Sources of Change

- New market conditions dictate changes to product requirements or business rules
- New customer needs demand modification of data, functionality, or services

- Business reorganization causes changes in project priorities or SE team structure
- Budgetary or scheduling constraints require system to be redefined

Topic 7.49 Change Requests

- Requests can come from users, customers, or management
- Change requests should be carefully analyzed as part of the maintenance process before they are implemented
- Some changes requests must be implemented urgently due to their nature
  - fault repair
  - system environment changes
  - urgently required business changes

Topic 7.50 Change Prediction

- Predicting the number of changes requires understanding the relationships between a system and its environment
- Tightly coupled systems require changes whenever the environment changes
- Factors influencing the system/environment relationship
  - number and complexity of system interfaces
  - number and volatility of system requirements
  - business processes where the system is uses

Topic 7.51 Configuration Management Tasks

- Identification
  - tracking changes to multiple SCI versions
- Version control
  - controlling changes before and after customer release
- Change control
  - authority to approve and prioritize changes
- Configuration auditing
  - ensure changes are made properly
- Reporting
  - tell others about changes made

Topic 7.52 Version Control Terms

- Entity
  - composed of objects at the same revision level
- Variant

  - a different set of objects at the same revision level and coexists with other variants
- New version
  - defined when major changes have been made to one or more objects

Topic 7.53 Change Control Process - 1

- Change request is submitted and evaluated to assess its technical merit and impact on the other configuration objects and budget
- Change report containing the results of the evaluation is generated
- Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report

Topic 7.54 Change Control Process - 3

- Modified object is checked-in to the project database and version control mechanisms are used to create the next version of the software
- Synchronization control is used to ensure that parallel changes made by different people don't overwrite one another

Topic 7.55 Configuration Management Team

- Analysts.
- Programmers.
- Program Librarian.

Topic 7.56 Change Control Board

- Customer representatives.
- Some members of the Configuration management team.

Topic 7.57 Programmer's View – 1

- Problem is discovered.
- Problem is reported to configuration control board.
- The board discusses the problem
  - is the problem a failure?
  - is it an enhancement?
  - who should pay for it?
- Assign the problem a priority or severity level, and assign staff to fix it.

Topic 7.58 Programmer's View - 2

- Programmer or analyst
  - locates the source of the problem
  - determines what is needed to fix it

- Programmer works with the librarian to control the installation of the changes in the operational system and the documentation.
- Programmer files a change report documenting all changes made.

Topic 7.59 Change Control Issues

- Synchronization (when?)
- Identification (who?)
- Naming (what?)
- Authentication (done correctly?)
- Authorization (who O.K.'d it?)
- Routing (who's informed?)
- Cancellation (who can stop it?)
- Delegation (responsibility issue)
- Valuation (priority issue)

Topic 7.60 Software Configuration Audit - 1

- Has the change specified by the ECO been made without modifications?
- Has an FTR been conducted to assess technical correctness?
- Was the software process followed and software engineering standards applied?

Topic 7.61 Software Configuration Audit – 2

- Do the attributes of the configuration object reflect the change?
- Have the SCM standards for recording and reporting the change been followed?
- Were all related SCI's properly updated?

Topic 7.62 Configuration Status Report

- What happened?
- Who did it?
- When did it happen?
- What else will be affected by the change?

8. CS 13 & 14

Topic 8.1 Test Process Improvement

- TPI (Test Process Improvement) is a structured and systematic approach used to enhance the quality and effectiveness of software testing within an organization.
- TPI focuses on identifying weaknesses and areas for improvement in the testing process, tools, and resources, with the ultimate goal of improving the overall testing capability and software quality.
- Improvement of the test process from various standpoints.

Topic 8.2 standpoints.

- **Identifying defects**: Testing is about finding and fixing defects, such as bugs, missing requirements, and incorrect functionality.
- **Focusing on high-risk areas**: It's not possible to test every combination of scenarios in a software application, so testers should focus on the most critical areas.
- **Testing early**: It's more cost-effective to identify and fix defects early in the development process.
- **Verifying requirements**: Testing ensures that the product meets the functional, performance, design, and implementation requirements.
- **Providing information**: Testing can help determine if a product is ready for market.
- **Increasing reliability**: Testing can help identify and fix bugs, making the software more dependable.
- **Improving performance**: Testing can highlight areas for improvement and performance enhancements.
- **Ensuring user-focus**: Testing can help align the software with evolving user needs and feedback.
- **Enabling compatibility**: Testing can ensure that the software performs well across different platforms, devices, and environments.
- An improved process is not only more capable of finding better bugs but also testing the application in ways that uplift its quality and standards.

Topic 8.3 When to Perform Test Process Improvement?

- **Significant rise in bugs**
- **Increase in complexity (test management)**
- **Increase in involved resources**
- **Increase in time of testing**
- **Increase in testing costs**
- **Newer methods have arrived**
- **not retrospected for a long time**

Topic 8.4 Benefits of Test Process Improvement

- **Testing improvements**
- **Good quality software**
- **Align testing with other phases**
- **Enhanced Value of Testing**
- **Reduced Downtime and Minimized Mistakes**
- **Adherence to Industry Standards**
- **Sustainable Cost Savings**
- **Accelerated Project Schedules**
- **Effective Evaluation of Improvement Efforts**

Topic 8.5 Implement Test Process Improvement

- Test process improvement is a process of analysis and acting upon our observation.
- Implementation process can be divided into four parts.
  - **Diagnose the test improvement**
  - **Initiate the process/planning phase**
  - **Acting on the plan**
  - **Verify, Report, and Learn**

Topic 8.6 Diagnose the test improvement

Topic 8.7 Diagnose the situation :

- Determining why we need to perform test improvement
- Report one or more things discussed in the previous section
- Once we report our findings, we need to dig a little deeper into those areas to explore specific findings that can point us to the exact problems.
- For example, if our concern is cost management, then, we need to document what the costs were earlier, how it has grown over time, graphs depicting a clear view of the growth, what has impacted this high cost in the recent past, any significant change in the resource, etc. Consider this document as the sole evidence of moving further with test process improvement.

Topic 8.8 Initiate the process/planning phase

- Plan the actions we will take for the process improvement.
- Includes documenting our objectives (such as budget control with reference to the previous example), goals, processes we need to follow with steps, and coverage of process improvement.
- we also need to put elements such as risks that can slow down our process or even force us to halt before completion. If any risk is identified, we can put down the expected delay in days or hours depending on the risk.

- Finally, at last, we document the final expected date of completion of the test process improvement and the date (or release version) from which we will start seeing the results of our endeavour.

Topic 8.9 Acting on the plan

- With all the blueprints in our hand, we start the actual method of test process improvement.
- This should follow the guidelines and expectations described during the planning phase and focus only on defects pointed out during the diagnosis phase.
- The senior testers are required to monitor and guide this process through their expertise.
- They should also ensure that delivery dates are not impacted and the process remains on schedule.

Topic 8.10 Verify, Report, and Learn

- At last, we verify the actions we performed with the results we got and match them with the expectations we set during the planning phase.
- Next, we report all our results (including metrics) on the report for approval and sign-off from senior testers and higher management.
- Lastly, we retrospect on the test improvement process from phases 1 to 4 and document our learnings throughout the process.

Topic 8.11 Measure Testing Process Improvement Impact

- **Define Clear Objectives:** Clearly define the objectives of the testing process improvement.
- **Establish Baseline Metrics:** Before implementing improvements, establish baseline metrics to understand the current state of the testing process.
- **Implement Changes Gradually:** Introduce process improvements gradually rather than all at once. This allows for a more controlled assessment of the impact of each change on the overall testing process.
- **Monitor Key Performance Indicators (KPIs):** Identify key performance indicators that directly align with the objectives of the testing process improvement.
- **Compare Metrics Before and After Implementation** Compare the baseline metrics with data collected after the implementation of testing process improvements.
- **Calculate Return on Investment (ROI):** Evaluate the return on investment by comparing the benefits gained from testing process improvements against the costs incurred in implementing those changes.
- **Utilize Surveys and Feedback:** Administer surveys or conduct feedback sessions with the testing team to gauge their perceptions of the impact of process improvements.

Topic 8.12 Software Testing Process Improvement Models

- Similar to software development models (such as Agile and Waterfall), we rely on many test process improvement models to follow a set standard and improve our testing process.
- These models are often divided into “maturities”
- which means a certain stage that progresses toward better arrangement, organization, and completion.

Topic 8.13 Testing Maturity Model integration (TMMi)

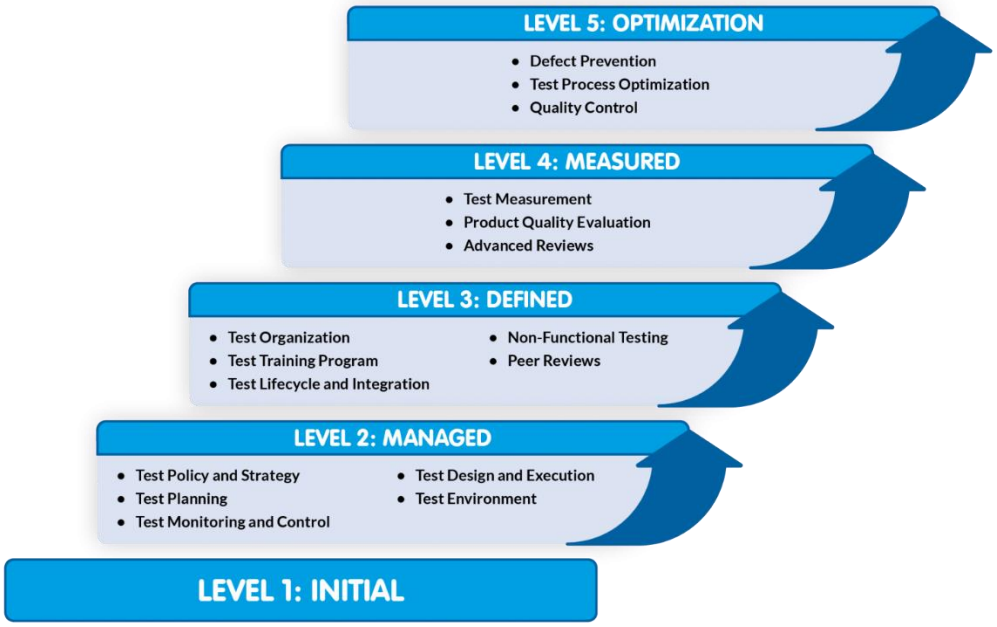
- TMMi is a “*maturity model*” which means it focuses on the maturity levels of each process to effectively implement the improvement in the testing lifecycle.

- Through TMMi, we can determine the maturity level of the testing process in an organization and improve it to achieve higher maturity.
- It was introduced in 2005 as a response to the ineffectiveness of the **Capability Maturity Model (CMM)** in the testing domain.
- TMMi tries to involve the organization structure and all the testers to progress together in the improvement process.

The benefits of using TMMi models are:

- Covers test-related activities extensively.
- Covers the best practices from the existing models.
- Covers the requirements for current trends and needs.
- Covers all aspects of test quality as described by industry experts.
- Provides a common standard for use.
- Can be applied to all phases of the software development lifecycle.

TMMi is divided into five maturity levels.



Topic 8.14 Maturity Level 1 – Initial

Maturity level 1 is not defined by the TMMi model because it presumes that all organizations start at this basic level. So, whatever you currently do and whatever process you follow, you can assume it to be of maturity level 1. This includes debugging and chaotic (unorganized) processes.

Topic 8.15 Maturity Level 2 – Managed

If the organization follows even a basic test approach towards testing and manages the same approach, it comes to maturity level 2. At this level, the organization should have a foundational structure in place that includes test planning, test policy, monitoring, and setting up of a test environment. This, however, depends deeply on the project as a lot of things change when a new project is introduced.

Topic 8.16 Maturity Level 3 – Defined

Maturity level 3 standardizes the testing process across the organization and expects each project to follow the same process. With this level, testing is now integrated very early into the development making it part of the development cycle.

Teams are required to be trained in testing and each member has a specific job for testing i.e. teams are more organized than before.

Maturity level 3 also expects that non-functional testing be planned and executed accordingly with reviews.

Topic 8.17 Maturity Level 4 – Measured

When the outcomes and parameter values are applied to all the projects to ensure a bug-free application, we reach maturity level 4. The review practices introduced in the maturity level 3 are now more advanced and thorough in nature.

Topic 8.18 Maturity Level 5 – Optimization

At maturity level 5, the organization has a series of methods set up for optimization of the processes followed in testing up to maturity level 4. Continuous optimization leads to a bug-free application.

Topic 8.19 Test Process Improvement (TPI) Next

- The Test Process Improvement Next model, also called TPI Next, is the next generation of its predecessor TPI (Test Process Improvement) model.
- TPI Next aims to strongly establish its predecessor's strengths while keeping in focus the business goals of the organization.
- Since TPI Next aims at the business-related key areas, it can be referred to for activities beyond testing and is a generic model to implement.

TPI Next model describes four maturity levels:

Topic 8.20 Maturity Level 1 – Initial

Initial maturity level has the same meaning in TPI (Next) as in the TMMi model. There is no fixed process, the organization does not follow any rules and the process is chaotic. Here too it is assumed that the organization is already at the maturity level 1 with whatever approach they have been following.

Topic 8.21 Maturity Level 2 – Controlled

A controlled maturity level includes the stakeholders to streamline the process with minimum structure. The structure is organized and helps in reducing bugs. However, this just represents a controlled state of the process and in no way is optimized. Therefore, we can assume a lot of bugs will still exist if the organization is at maturity level 2.

Topic 8.22 Maturity Level 3 – Efficient

Maturity level 3 aims at the efficiency of the currently working process in testing. At this level, we make sure we are not spending extra time on the redundant processes that arise when the testing process has certain overlapping stages. We try to streamline these processes and combine each process for a unified testing cycle.

Topic 8.23 Maturity Level 4 – Optimizing

In the final level of maturity, we assign coordinators who evaluate the process continuously from time to time. Their evaluation helps optimize the process and improve it for the next project. This includes evaluating the bugs we faced, any delays, or any other type of hindrances faced during the run.

Topic 8.24 Systematic Test and Evaluation Process (STEP)

A STEP model approach focuses on testing the requirements systematically, evaluating the results, optimizing accordingly, and then moving forward.

The basic idea of the STEP model is to accompany testing along with other parts of SDLC and testing should be involved from the beginning to ensure good quality software.

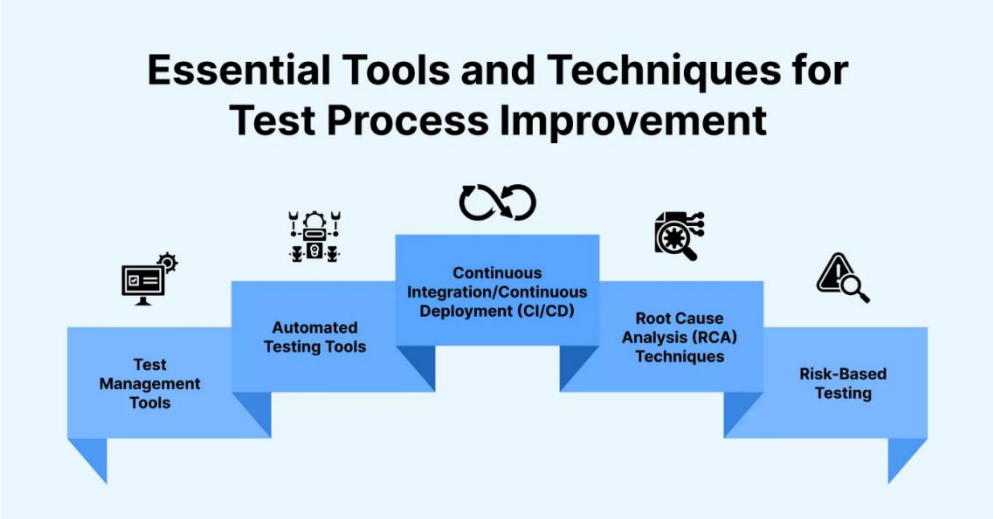
characteristics of STEP

STEP focuses on requirement-driven testing

STEP aims at performing testing first and then starting coding.

STEP helps in early defect detection due to the “testing first” approach.

Topic 8.25 Tools and Techniques for Test Process Improvement



Topic 8.26 Test Management Tools :

- To successfully improve the test process, test management must be effective.
- Teams can effectively plan, carry out, and monitor testing operations using test management solutions like Jira, TestRail, and HP ALM (Application Lifecycle Management).

Topic 8.27 Automated Testing Tools:

- Teams may automate time-consuming and repetitive test cases using tools like Selenium, Appium, and JUnit, freeing up testers to concentrate on more important scenarios.
- Automation broadens the scope of tests, quickens the testing process, and improves the accuracy of test results. Additionally, it makes early bug discovery easier, which lowers the expense and work needed to correct flaws.

Topic 8.28 Continuous Integration/Continuous Deployment (CI/CD):

- Implementing CI/CD pipelines guarantees that software be released more quickly and frequently without sacrificing quality.
- The processes for building, integrating, and deploying software are automated by tools like Jenkins, GitLab CI/CD, and CircleCI.
- Organizations may do continuous testing, identify issues early, and establish a smooth and reliable deployment process by integrating automated tests into the CI/CD pipeline.

Topic 8.29 CMMi

- The Capability Maturity Model Integration (CMMI) is a model that aids in identifying the strengths and weaknesses of an organization's current processes and shows the way to improvement.
- CMMI's primary goal is to create high-quality software.
- CMMI is a technology offered by SEI that assists businesses in standardizing **software development**, **testing**, and deployment to improve the product's quality.
- The implementation of standards that will assist raise the quality of their software products is supported by CMMI.
- According to CMMI, a complete model consisting of five "Maturity Levels" or three "Capability Levels" is essential to creating excellent software.
- CMMI is a more advanced model of its CMM predecessor.

Topic 8.30 What is CMM?

- CMM: Capability Maturity Model
- Developed by the Software Engineering Institute of the Carnegie Mellon University
- Framework that describes the key elements of an effective software process.
- Describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one.
- Provides guidance on how to gain control of processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence.

Topic 8.31 Process Maturity Concepts

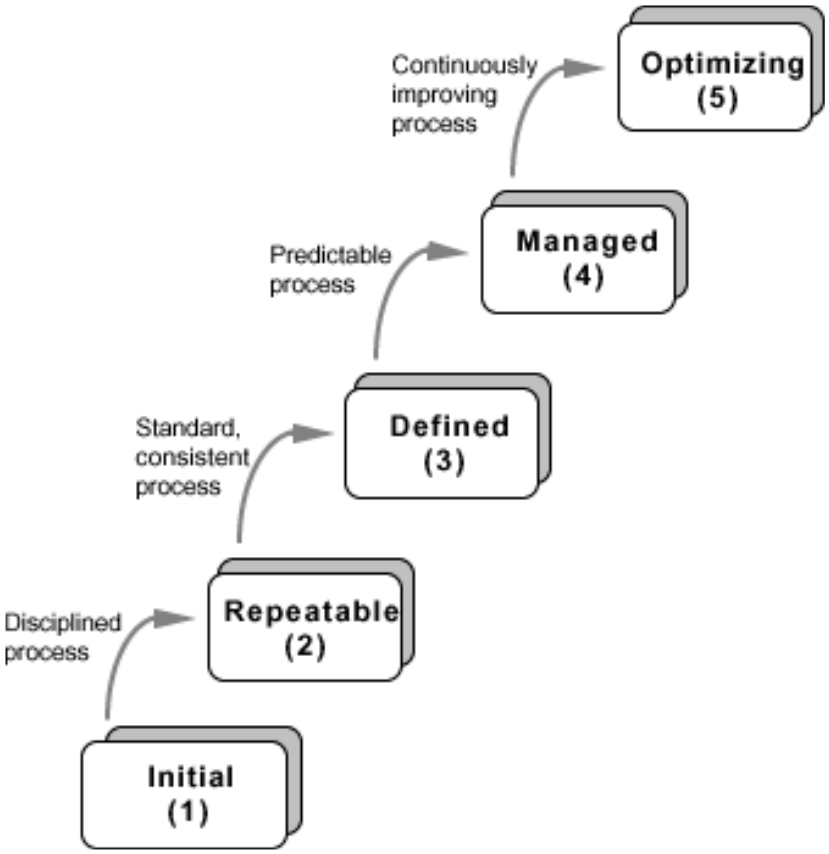
- Software Process
  - set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, user manuals)
- Software Process Capability
  - describes the range of expected results that can be achieved by following a software process
  - means of predicting the most likely outcomes to be expected from the next software project the organization undertakes

- Software Process Performance
  - actual results achieved by following a software process
- Software Process Maturity
  - extent to which a specific process is explicitly defined, managed, measured, controlled and effective
  - implies potential growth in capability
  - indicates richness of process and consistency with which it is applied in projects throughout the organization

Topic 8.32 What are the CMM Levels? (The five levels of software process maturity)

Maturity level indicates level of process capability:

- Initial
- Repeatable
- Defined
- Managed
- Optimizing



Topic 8.33 Level 1: Initial

- Initial : The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

- At this level, frequently have difficulty making commitments that the staff can meet with an orderly process
- Products developed are often over budget and schedule
- Wide variations in cost, schedule, functionality and quality targets
- Capability is a characteristic of the individuals, not of the organization

Topic 8.34 Level 2: Repeatable

- Basic process management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
  - Realistic project commitments based on results observed on previous projects
  - Software project standards are defined and faithfully followed
  - Processes may differ between projects
  - Process is disciplined
  - earlier successes can be repeated

Topic 8.35 Level 3: Defined

- The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing an maintaining software.

Topic 8.36 Level 4: Managed

- Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
  - Narrowing the variation in process performance to fall within acceptable quantitative bounds
  - When known limits are exceeded, corrective action can be taken
  - Quantifiable and predictable
    - predict trends in process and product quality

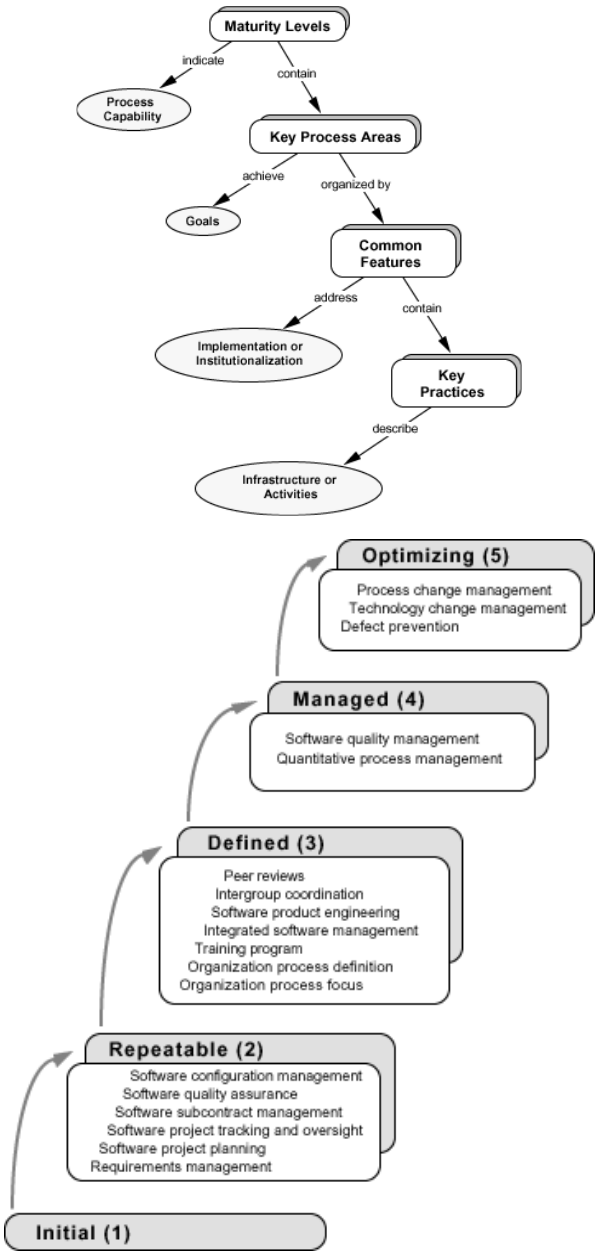
Topic 8.37 Level 5: Optimizing

- Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.
- Goal is to prevent the occurrence of defects
  - Causal analysis
- Data on process effectiveness used for cost benefit analysis of new technologies and proposed process changes

Topic 8.38 Internal Structure to Maturity Levels

- Except for level 1, each level is decomposed into key process areas (KPA)
- Each KPA identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing software capability.
  - commitment
  - ability
  - activity
  - measurement

- verification



Topic 8.39 Level 2 KPAs

- Requirements Management
  - Establish common understanding of customer requirements between the customer and the software project
  - Requirements is basis for planning and managing the software project
  - Not working backwards from a given release date!
- Software Project Planning
  - Establish reasonable plans for performing the software engineering activities and for managing the software project

- Software Project Tracking and Oversight
  - Establish adequate visibility into actual progress
  - Take effective actions when project's performance deviates significantly from planned
- Software Subcontract Management
  - Manage projects outsourced to subcontractors
- Software Quality Assurance
  - Provide management with appropriate visibility into
    - process being used by the software projects
    - work products
- Software Configuration Management
  - Establish and maintain the integrity of work products
  - Product baseline
  - Baseline authority

Topic 8.40 Level 3 KPAs

- Organization Process Focus
  - Establish organizational responsibility for software process activities that improve the organization's overall software process capability
- Organization Process Definition
  - Develop and maintain a usable set of software process assets
    - stable foundation that can be institutionalized
    - basis for defining meaningful data for quantitative process management
- Training Program
  - Develop skills and knowledge so that individual can perform their roles effectively and efficiently
  - Organizational responsibility
  - Needs identified by project
- Integrated Software Management
  - Integrated engineering and management activities
  - Engineering and management processes are tailored from the organizational standard processes
  - Tailoring based on business environment and project needs
- Software Product Engineering
  - technical activities of the project are well defined (SDLC)

  - correct, consistent work products
- Intergroup Coordination
  - Software engineering groups participate actively with other groups
- Peer Reviews
  - early defect detection and removal
  - better understanding of the products
  - implemented with inspections, walkthroughs, etc

Topic 8.41 Level 4 KPAs

- Quantitative Process Management
  - control process performance quantitatively
  - actual results from following a software process
  - focus on identifying and correcting special causes of variation with respect to a baseline process
- Software Quality Management
  - quantitative understanding of software quality
    - products
    - process

Topic 8.42 Level 5 KPAs

- Process Change Management
  - continuous process improvement to improve quality, increase productivity, decrease cycle time
- Technology Change Management
  - identify and transfer beneficial new technologies
    - tools
    - methods
    - processes
- Defect Prevention
  - causal analysis of defects to prevent recurrence

Topic 8.43 What are the benefits ?

- Helps forge a shared vision of what software process improvement means for the organization
- Defines set of priorities for addressing software problems
- Supports measurement of process by providing framework for performing reliable and consistent appraisals

- Provides framework for consistency of processes and product

Topic 8.44 Why measure software and software process?

Obtain data that helps us to better control

- schedule
- cost
- quality of software products

Topic 8.45 Consistent measurement provide data for:

- Quantitatively expressing requirements, goals, and acceptance criteria
- Monitoring progress and anticipating problems
- Quantifying tradeoffs used in allocating resources
- Predicting schedule, cost and quality

Topic 8.46 Measurements

- Historical
- Plan
- Actual
- Projections

Topic 8.47 SEI Core Measures

Unit of Measure	Characteristics Addressed
Physical source lines of code Logical source lines of code	Size, reuse, rework
Staff hours	Effort, cost, resource allocations
Calendar dates for process milestones Calendar dates for deliverables	Schedule, progress
Problems and defects	Quality, improvement trends, rework, readiness for delivery

Topic 8.48 Examples of measurements for size of work products

- Estimated number of requirements
- Actual number of requirements
- Estimated source lines of code (SLOC)
- Actual SLOC
- Estimated number of test cases
- Actual number of test cases

Topic 8.49 Example of measurements of effort

- Estimated man-hours to design/code a given module
- Actual man-hours expended for designing/coding the module
- Estimated number of hours to run builds for a given release
- Actual number of hours spent running builds for the release

Topic 8.50 Examples of measurements of quality of the work product

- Number of issues raised at requirements inspection
- Number of requirements issues open
- Number of requirements issues closed
- Number of issues raised during code inspection
- Number of defects opened during unit testing

Topic 8.51 Examples of measurements of quality of the work product

- Number of defects opened during system testing
- Number of defects opened during UAT
- Number of defects still open
- Number of defects closed
- Defect age

Topic 8.52 Examples of measurements of quality of the work product

- Total number of build failures
- Total number of defects fixed for a given release
- Total number of defects verified and accepted
- Total number of defects verified and rejected

Topic 8.53 levels of CMMI

- An organization receives one of two ratings following a Class A appraisal: a maturity level rating or a capacity level rating.
- Maturity levels range from 1 to 5, with level 5 as the highest grade and the target businesses aim towards.

Topic 8.54 1. Initial Level

- The processes at this CMMI level tend to be erratic and reactive. The organization is at its worst at this point due to the unpredictability of the environment and the likelihood of errors and ineptitude.

Topic 8.55 2. Managed processes

- At maturity level #2, an organization has completed all the process areas' specialized and general goals. In other words, the organization's initiatives have ensured that processes are planned, carried out, measured, and controlled.

Standards implemented at this level are usually as follows:

- Requirements Management, or REQM
- Project Planning (PP)
- Configuration Management, or CM
- Measurement and Analysis (MA)
- Process and Product Quality Assurance (PPQA)
- Project Monitoring and Control, or PMC
- Supplier Agreement Management (SAM).

Topic 8.56 3. Defined processes

- Organizations take a more preventative approach than a reactive one at this level.
- Managers are now aware of the flaws and how to fix them to enhance their operations. There are several KPAs of which helps to offer direction across projects, initiatives, and portfolios:
  - Decision Analysis and Resolution, or DAR.
  - Organizational Process Focus (OPF)
  - Integrated Project Management (IPM) plus IPPD (Integrated Product and Process Development)
  - OPD stands for organizational process definition, while OT stands for executive training.
  - Product Integration (PI)
  - Risk Management: RSKM
  - Validation, or VAL
  - Technical Solution (TS)
  - Verification, VER

Topic 8.57 4. Managed quantitatively

- The company has reached a high maturity level and relies on predictable methods based on the stakeholders' needs.
- The procedures are better organized, respectable, and exact. The firm anticipates risks and uses a data-driven strategy to address process flaws, and it is reflected in the following KPAs :
    - OPP – Organizational Process Performance
    - QPM – Quantitative Project Management

Topic 8.58 5. Optimizing

- The organization is currently in a stable and adaptable phase. Now, the company is always striving for progress and seizing possibilities.
- In an expected environment, the company pursues "agility and innovation" at CMMI level 5, also known as the optimizing level.

- Organizations reach a high degree of maturity when they reach Levels 4 and 5, where they are continually evolving to satisfy the demands of their clients and investors.

Topic 8.59 Six Sigma in Software Engineering

- Six Sigma is a methodology that helps organizations in making their process better and more efficient by identifying and removing errors and variations.
- Variations in processes can lead to errors, these errors can lead to product defects and product defects can lead to poor customer satisfaction.
- By reducing variation and errors Six Sigma can reduce process costs and increase customer satisfaction.
- It is a statistical concept that aims to define the variation found in any process.
- Six Sigma is a process of producing high and improved quality output. This can be done in two phases – identification and elimination.
- The cause of defects is identified and appropriate elimination is done, which reduces variation in whole processes.
- Six Sigma processes have a failure rate of only 3.4 per million opportunities i.e. 99.99966 percent of Six Sigma products are free from defect, while Five Sigma processes have a failure rate of only 233 errors per million opportunities.

Topic 8.60 Characteristics of Six Sigma

- **Statistical Quality Control:** Six Sigma is derived from the Greek Letter ? which denote Standard Deviation in statistics. Standard Deviation is used for measuring the quality of output.
- **Methodical Approach:** The Six Sigma is a systematic approach of application in DMAIC and DMADV which can be used to improve the quality of production. DMAIC means for Design-Measure- Analyze-Improve-Control. While DMADV stands for Design-Measure-Analyze-Design-Verify.
- **Fact and Data-Based Approach:** The statistical and methodical method shows the scientific basis of the technique.
- **Project and Objective-Based Focus:** The Six Sigma process is implemented to focus on the requirements and conditions.
- **Customer Focus:** The customer focus is fundamental to the Six Sigma approach. The quality improvement and control standards are based on specific customer requirements.
- **Teamwork Approach to Quality Management:** The Six Sigma process requires organizations to get organized for improving quality.

Topic 8.61 The 6 Key Principals of Six Sigma

Topic 8.62 Customer Centric Improvement

- The primary principal of six sigma methodology is to focus on customer.
- Voice of the Customer (VoC) and methods for determining what the customer truly want from a product or process. Organizations can boost customer happiness by combining that knowledge with measurements, analytics, and process improvement approaches, resulting in higher profits, client retention, and loyalty.

Topic 8.63 Continuous Process Improvement

- The Six Sigma approach requires constant process improvement. An organization that fully implements the Six Sigma technique never stops improving.
- It continuously discovers and priorities opportunities. Once one area has been improved, the organization will move on to another.
- The organization continuously find ways to increase the sigma level because the goal is to achieve the level of 99.99966 accuracy for all processes inside an organization while also making sure other essentials like financial stability.

Topic 8.64 Reduce Variation

- A method to continuously improve a process is to reduce its variation. Every process has an inherit variation. Variation in processes can lead to errors, these errors can lead to product defect and product defect can leads to poor customer satisfaction.
- By reducing variation and errors six sigma can reduce process cost and increase customer satisfaction. Suppose there are some developers developing web application, variation will exist as every developer have different coding styles, expertise levels, environment factors and project requirement.
- By adopting strategies like coding standards and guideline, code reviews, automated testing and documentation variation can be reduce to some extent.

Topic 8.65 Eliminating Waste

- Waste is a major problem in the six sigma methodology. Eliminating waste means removing items, procedures or people that are not required for the process's outcome or removing

anything that does not add value to customer. Eliminating waste can reduce processing time, errors in process and lowers overall costs.

Topic 8.66 Empowering Employees

- Until organizations provide employees with the tools they need to monitor and sustain improvements, implementing improved processes is only a temporary solution.
- Process improvement usually involves two approaches in most organizations.
- An improvement is first defined, planned, and carried out by a process improvement team consisting up of project managers, methodology specialists, and subject matter experts.
- The employees that deal with the process on a daily basis are then equipped by that team to supervise and handle it in its improved condition.

Topic 8.67 Controlling the Process

- Six Sigma improvements are frequently used to handle uncontrolled processes. Out-of-control processes meet certain statistical conditions.
- The purpose of improvement is to bring a process back under statistical control.
- Then, after the improvements are implemented, measurements, statistics, and other Six Sigma tools are utilized to keep the process under control. Implementing controls and training people on how to apply them is a key component of continuous improvement.

Topic 8.68 The Six Sigma Methodology

The Two Six Sigma methodologies used in the Six Sigma projects are DMAIC and DMADV.

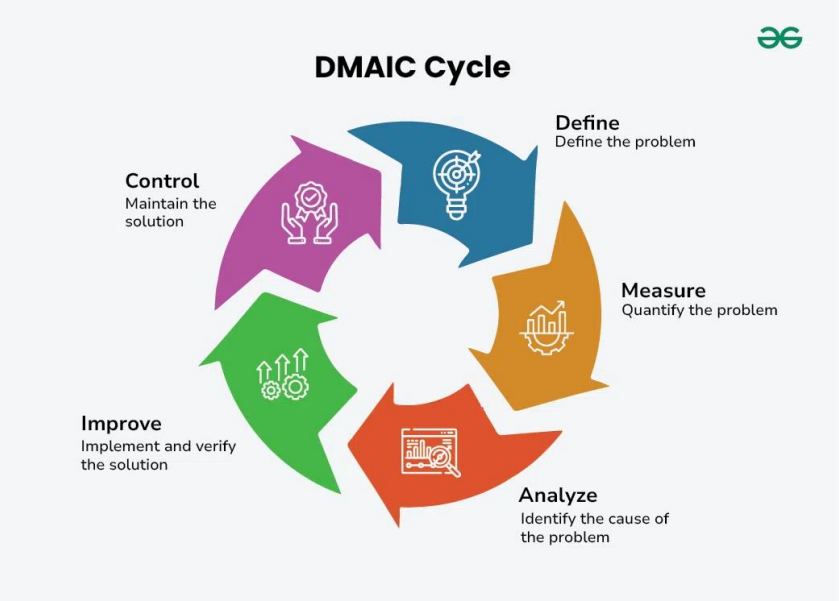
Six Sigma teams usually use DMAIC or DMADV approaches to achieve process improvements and establish process control.

Topic 8.69 DMAIC Six Sigma Methodology

DMAIC is used to enhance an existing business process.

A DMAIC project involves identifying important problem that are creating the problem, verifying those problem , brainstorming solutions, implementing them, and designing a control plan to maintain the improved state.

The DMAIC methodology is designed for the team who are responsible for improving a project.



The DMAIC project methodology has five phases:

- Define
- Measure
- Analyze
- Improve
- Control

Topic 8.70 Define

The Define phase of a DMAIC project involves identifying problems, establishing project requirements, and setting success goals. Six Sigma leaders can use tools inside the phase to create flexibility for different project types, depending on factors such as leadership advice and budgets.

Topic 8.71 Measure

During the DMAIC Measure phase, teams use data to validate assumptions about the process and problem. Validation of assumptions also makes it into the analysis step.

The measurement phase focuses on collecting and arranging data for analysis. Measuring in a Six Sigma project might be challenging without proper data collection. To gather data, teams may need to build tools, create queries, filter through large amounts of information, or use manual processes.

Topic 8.72 Analyze

Analyze phase is a critical stage where the root causes of problems or inefficiencies within a process are identified and understood.

During the Analyze phase of a DMAIC project, teams develop predictions about relationships between inputs and outputs, use statistical analysis and data to validate the prediction and assumptions they've made thus far.

In a DMAIC project, the Analyze phase leads to the Improve phase, where hypothesis testing can confirm assumptions and potential solutions.

Topic 8.73 Improve

During the Improve phase of a project, Six Sigma teams begin developing the concepts that came from the Analyze phase.

They employ statistics and real-world observations to test assumptions and solutions.

As teams select and start implementing solutions, hypothesis testing keeps going through

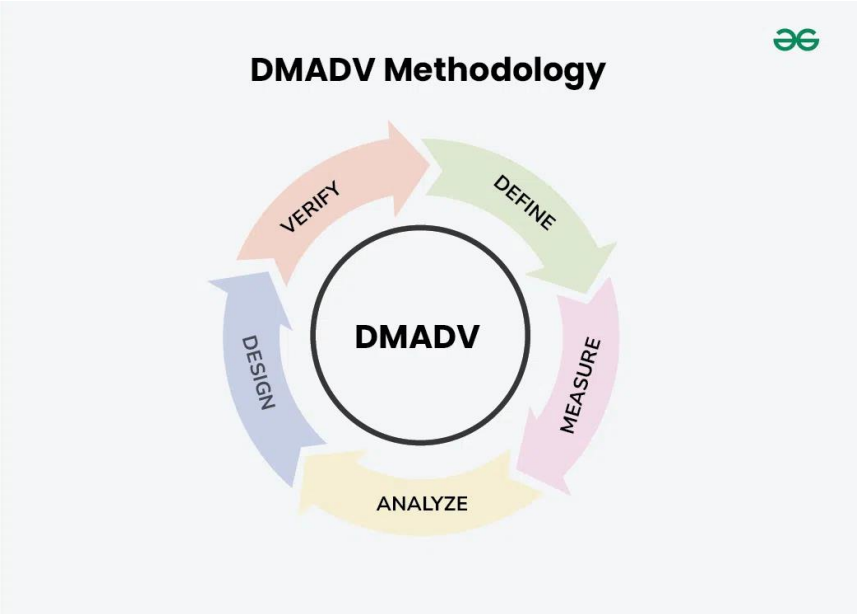
Topic 8.74 Control

In DMAIC Phase Controls and standards are established so that improvements can be maintained, but the responsibility for those improvements is transitioned to the process owner.

Topic 8.75 DMADV Six Sigma Methodology

DMADV is used to create new product designs or process designs. Six Sigma teams use DMADV in the following scenario:

- The organization wants to launch a new service or product.
- Business leaders decide to replace a process to meet upgrade requirements or to align business processes, machinery, or workers with future goals.
- A Six Sigma team learns that upgrading a process is unlikely to result in the expected project outcomes.



The DMADV project methodology also has **five** phases:

- Define
- Measure
- Analyze
- Design
- Verify

Topic 8.76 Define

In a DMADV project, the Define stage is slightly more strict. Teams must identify problems and define requirements within a change management environment. When an organization has a change management program in place, Six Sigma teams must include all program needs in the DMADV stages.

Topic 8.77 Measure

During the DMADV Measure phase, teams use data to validate assumptions about the process and problem. Validation of assumptions also makes it into the analysis step. The measurement phase focuses on collecting and arranging data for analysis.

Topic 8.78 Analyze

Analyze phase is a critical stage where the root causes of problems or inefficiencies within a process are identified and understood. They priorities identifying best practices and standards for measuring and designing new processes.

Topic 8.79 Design

The fourth phase is when DMADV projects start to vary significantly from DMAIC projects. The team designs a new process that includes solution testing, mapping, workflow principles, and infrastructure development.

Topic 8.80 Verify Phase

The Verify phase in DMADV checks if the designed solutions work as intended, measuring their success against initial goals, ensuring improvements are effective and sustainable.

Topic 8.81 Difference between DMAIC and DMADV

The primary difference between DMAIC and DMADV in terms of team goals and project outcomes.

Both methodology aims to deliver better quality, better efficiency, more production, more profits and provide excellent customer satisfaction.

Topic 8.82 Conclusion- six sigma

Six Sigma is a structured methodology used by organization to improve processes by reducing inherit variation and defects.

Six Sigma helps the organization in improving the efficiency, quality and customer satisfaction by reducing variation and defects in processes.

Six Sigma consists of two methodology DMAIC and DMADV.

DMAIC stands for “D-Define”, “M-Measure”, “A-Analysis”, “I-Improve”, “C-Control” . DMAIC is used to enhance an existing business process.

DMADV stands for “D-Define”, “M-Measure”, “A-Analysis”, “D-Design”, “V-Verify”. DMADV is used to create new product designs or process designs.

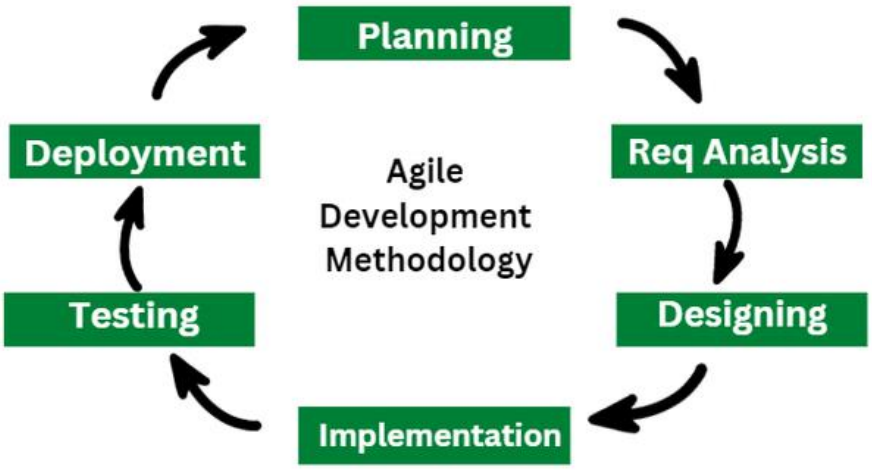
9. CS14 Software Quality Assurance and TestingModule 9

Topic 9.1 Introduction to Agile Methodology and Testing

Agile is a project management and software development approach that aims to be more effective.

- It focuses on delivering smaller pieces of work regularly instead of one big launch.
- This allows teams to adapt to changes quickly and provide customer value faster.

Topic 9.2 Life cycle of Agile Methodology



Topic 9.3 Agile Software Testing

Agile Testing is a type of software testing that follows the principles of agile software development to test the software application.

All members of the project team along with the special experts and testers are involved in agile testing.

Agile testing is not a separate phase and it is carried out with all the development phases i.e. requirements, design and coding, and test case generation.

Agile testing takes place simultaneously throughout the Development Life Cycle.

Agile testers participate in the entire development life cycle along with development team members and the testers help in building the software according to the customer requirements and with better design and thus code becomes possible.

The agile testing team works as a single team towards the single objective of achieving quality.

Agile Testing has shorter time frames called iterations or loops. This methodology is also called the delivery-driven approach because it provides a better prediction on the workable products in less duration time.

- In Agile, testing is an ongoing activity rather than a distinct phase, focusing on collaboration between testers, developers, and stakeholders.

- This approach helps detect and fix issues early, aligning with Agile's iterative development cycle.

Topic 9.4 Agile Testing Principles

Shortening feedback iteration: In Agile Testing, the testing team gets to know the product development and its quality for each and every iteration. Thus continuous feedback minimizes the feedback response time and the fixing cost is also reduced.

Testing is performed alongside Agile testing is not a different phase. It is performed alongside the development phase. It ensures that the features implemented during that iteration are actually done. Testing is not kept pending for a later phase.

Involvement of all members: Agile testing involves each and every member of the development team and the testing team. It includes various developers and experts.

Documentation is weightless: In place of global test documentation, agile testers use reusable checklists to suggest tests and focus on the essence of the test rather than the incidental details. Lightweight documentation tools are used.

Clean code: The defects that are detected are fixed within the same iteration. This ensures clean code at any stage of development.

Constant response: Agile testing helps to deliver responses or feedback on an ongoing basis. Thus, the product can meet the business needs.

Customer satisfaction: In agile testing, customers are exposed to the product throughout the development process. Throughout the development process, the customer can modify the requirements, and update the requirements and the tests can also be changed as per the changed requirements.

Test-driven: In agile testing, the testing needs to be conducted alongside the development process to shorten the development time. But testing is implemented after the implementation or when the software is developed in the traditional process.

Topic 9.5 Agile Testing Methodologies

Test-Driven Development (TDD): TDD is the software development process relying on creating unit test cases before developing the actual code of the software. It is an iterative approach that combines 3 operations, programming, creation of unit tests, and refactoring.

Behavior Driven Development (BDD): BDD is agile software testing that aims to document and develop the application around the user behavior a user expects to experience when interacting with the application. It encourages collaboration among the developer, quality experts, and customer representatives.

Exploratory Testing: In exploratory testing, the tester has the freedom to explore the code and create effective and efficient software. It helps to discover the unknown risks and explore each aspect of the software functionality.

Acceptance Test-Driven Development (ATDD): ATDD is a collaborative process where customer representatives, developers, and testers come together to discuss the requirements, and potential pitfalls and thus reduce the chance of errors before coding begins.

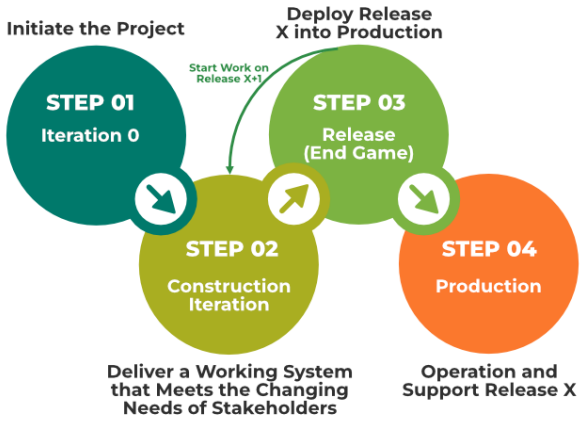
Extreme Programming (XP): Extreme programming is a customer-oriented methodology that helps to deliver a good quality product that meets customer expectations and requirements.

Session-Based Testing: It is a structured and time-based approach that involves the progress of exploratory testing in multiple sessions. This involves uninterrupted testing sessions that are time-boxed with a duration varying from 45 to 90 minutes. During the session, the tester creates a document called a charter document that includes various information about their testing.

Dynamic Software Development Method (DSDM): DSDM is an agile project delivery framework that provides a framework for building and maintaining systems. It can be used by users, developers, and testers.

Crystal Methodologies: This methodology focuses on people and their interactions when working on the project instead of processes and tools. The suitability of the crystal method depends on three dimensions, team size, criticality, and priority of the project.

Topic 9.6 Agile Testing Strategies



Topic 9.7 Iteration 0

- It is the first stage of the testing process and the initial setup is performed in this stage. The testing environment is set in this iteration.
    - This stage involves executing the preliminary setup tasks such as finding people for testing, preparing the usability testing lab, preparing resources, etc.
    - The business case for the project, boundary situations, and project scope are verified.
    - Important requirements and use cases are summarized.
    - Initial project and cost valuation are planned.
    - Risks are identified.
    - Outline one or more candidate designs for the project.

Topic 9.8 Construction Iteration

It is the second phase of the testing process. It is the major phase of the testing and most of the work is performed in this phase. It is a set of iterations to build an increment of the solution. This process is divided into two types of testing:

Confirmatory testing: This type of testing concentrates on verifying that the system meets the stakeholder's requirements as described to the team to date and is performed by the team. It is further divided into 2 types of testing:

- **Agile acceptance testing:** It is the combination of acceptance testing and functional testing. It can be executed by the development team and the stakeholders.
- **Developer testing:** It is the combination of unit testing and integration testing and verifies both the application code and database schema.

Investigative testing: Investigative testing detects the problems that are skipped or ignored during confirmatory testing.

In this type of testing, the tester determines the potential problems in the form of defect stories. It focuses on issues like integration testing, load testing, security testing, and stress testing.

Topic 9.9 Release End Game

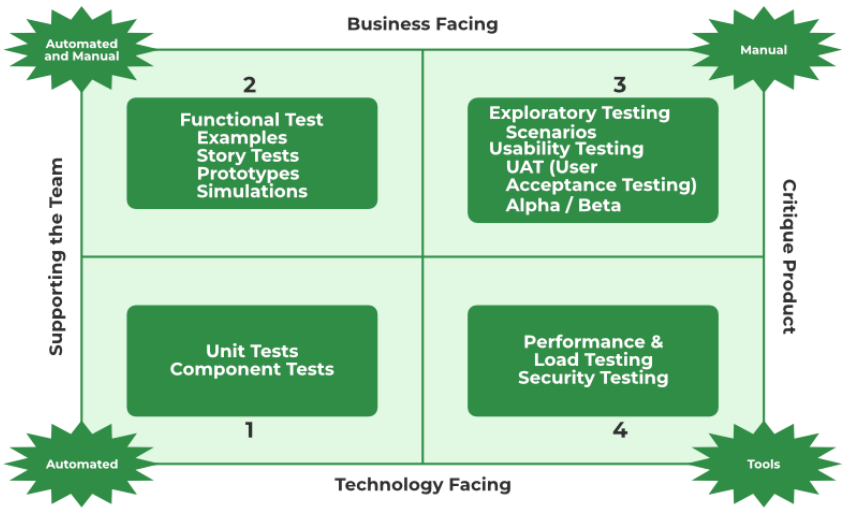
This phase is also known as the transition phase. This phase includes the full system testing and the acceptance testing. To finish the testing stage, the product is tested more relentlessly while it is in construction iterations. In this phase, testers work on the defect stories. This phase involves activities like:

- Training end-users.
- Support people and operational people.
- Marketing of the product release.
- Back-up and restoration.
- Finalization of the system and user documentation.

Topic 9.10 Production

It is the last phase of agile testing. The product is finalized in this stage after the removal of all defects and issues raised.

Topic 9.11 Agile Testing Quadrants



Topic 9.12 Quadrant 1 (Automated)

The first agile quadrat focuses on the internal quality of code which contains the test cases and test components that are executed by the test engineers.

All test cases are technology-driven and used for automation testing. All through the agile first quadrant of testing, the following testing can be executed:

- Unit testing.
- Component testing.

Topic 9.13 Quadrant 2 (Manual and Automated)

The second agile quadrant focuses on the customer requirements that are provided to the testing team before and throughout the testing process. The test cases in this quadrant are business-driven and are used for manual and automated functional testing. The following testing will be executed in this quadrant:

- Pair testing.
- Testing scenarios and workflow.
- Testing user stories and experiences like prototypes.

Topic 9.14 Quadrant 3 (Manual)

The third agile quadrant provides feedback to the first and the second quadrant.

This quadrant involves executing many iterations of testing, these reviews and responses are then used to strengthen the code. The test cases in this quadrant are developed to implement automation testing. The testing that can be carried out in this quadrant are:

- Usability testing.
- Collaborative testing.
- User acceptance testing.
- Collaborative testing.
- Pair testing with customers.

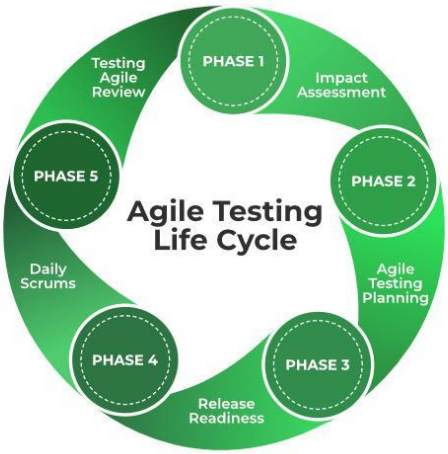
Topic 9.15 Quadrant 4 (Tools)

The fourth agile quadrant focuses on the non-functional requirements of the product like performance, security, stability, etc. Various types of testing are performed in this quadrant to deliver non-functional qualities and the expected value.

The testing activities that can be performed in this quadrant are:

- Non-functional testing such as stress testing, load testing, performance testing, etc.
- Security testing.
- Scalability testing.
- Infrastructure testing.
- Data migration testing.

Topic 9.16 Agile Testing Life Cycle



Topic 9.17 Agile Testing Life Cycle :

5 different phases:

Impact Assessment: This is the first phase of the agile testing life cycle also known as the feedback phase where the inputs and responses are collected from the users and stakeholders. This phase supports the test engineers to set the objective for the next phase in the cycle.

Agile Testing Planning: In this phase, the developers, customers, test engineers, and stakeholders team up to plan the testing process schedules, regular meetings, and deliverables.

Release Readiness: This is the third phase in the agile testing lifecycle where the test engineers review the features which have been created entirely and test if the features are ready to go live or not and the features that need to be sent again to the previous development phase.

Daily Scrums: This phase involves the daily morning meetings to check on testing and determine the objectives for the day. The goals are set daily to enable test engineers to understand the status of testing.

Test Agility Review: This is the last phase of the agile testing lifecycle that includes weekly meetings with the stakeholders to evaluate and assess the progress against the goals.

Topic 9.18 Agile Test Plan

- An agile test plan includes types of testing done in that iteration like test data requirements, test environments, and test results.
- In agile testing, a test plan is written and updated for every release.

The Agile test plan includes the following:

- Test Scope.
- Testing instruments.
- Data and settings are to be used for the test.
- Approaches and strategies used to test.
- Skills required to test.
- New functionalities are being tested.

- Levels or Types of testing based on the complexity of the features.
- Resourcing.
- Deliverables and Milestones.
- Infrastructure Consideration.
- Load or Performance Testing.
- Mitigation or Risks Plan.

Topic 9.19 Benefits of Agile Testing

Saves time: Implementing agile testing helps to make cost estimates more transparent and thus helps to save time and money.

Reduces documentation: It requires less documentation to execute agile testing.

Enhances software productivity: Agile testing helps to reduce errors, improve product quality, and enhance software productivity.

Higher efficiency: In agile software testing the work is divided into small parts thus developer can focus more easily and complete one part first and then move on to the next part. This approach helps to identify minor inconsistencies and higher efficiency.

Improve product quality: In agile testing, regular feedback is obtained from the user and other stakeholders, which helps to enhance the software product quality.

Topic 9.20 Limitations of Agile Testing

Project failure: In agile testing, if one or more members leave the job then there are chances for the project failure.

Limited documentation: In agile testing, there is no or less documentation which makes it difficult to predict the expected results as there are explicit conditions and requirements.

Introduce new bugs: In agile software testing, bug fixes, modifications, and releases happen repeatedly which may sometimes result in the introduction of new bugs in the system.

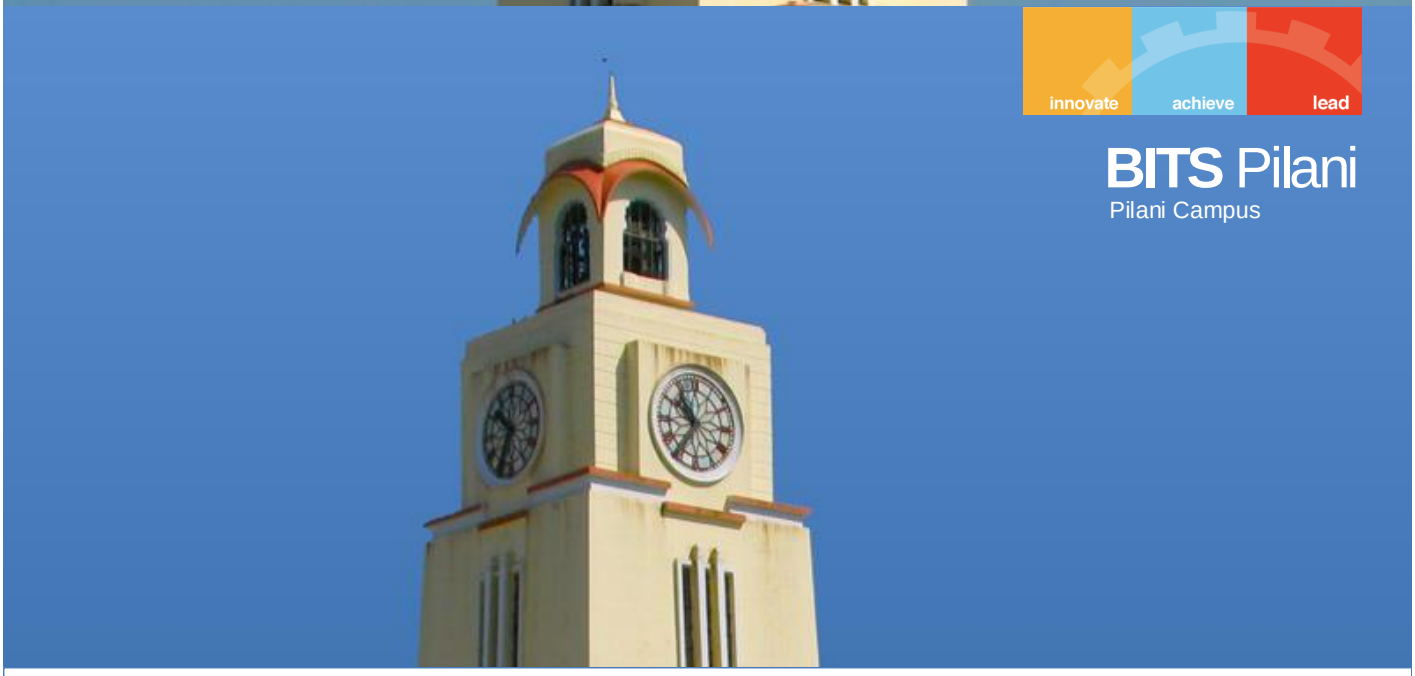
Poor planning: In agile testing, the team is not exactly aware of the end result from day one, so it becomes challenging to predict factors like cost, time, and resources required at the beginning of the project.

No finite end: Agile testing requires minimal planning at the beginning so it becomes easy to get sidetracked while delivering the new product. There is no finite end and there is no clear vision of what the final product will look like.

BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS



innovate achieve lead

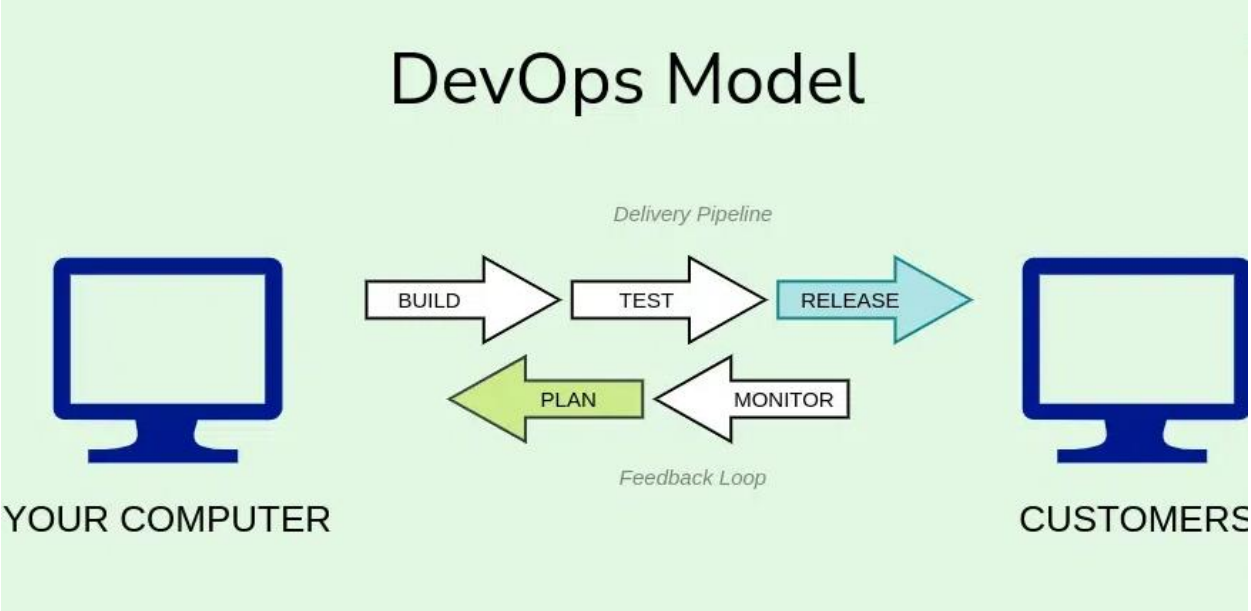
BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Module 9 ,10

DEVOPS

- As the technology landscape continues to shift, the demand for quicker and more reliable software delivery becomes even more imminent.
- Enter DevOps: an innovation designed to bridge the gap between software development and IT operations.

- **DevOps is a transformative culture and practice that unites software development (Dev) and IT operations (Ops) teams**.
- By fostering collaboration and leveraging automation technologies, DevOps enables faster, more reliable code deployment to production in an efficient and repeatable manner.
- DevOps is a software development approach that emphasizes collaboration and communication between development (Dev) and operations (Ops) teams.
- It aims to shorten the software development lifecycle and improve the quality and reliability of software releases.



innovate achieve lead

Delivery Pipeline

The pipeline represents the different stages that software goes through before it is released to production. These stages might typically include:

Build: The stage where the software code is compiled and packaged into a deployable unit.

Test: The stage where the software is rigorously tested to ensure it functions as expected and identifies any bugs.

Release: The stage where the software is deployed to production for end users.

Feedback Loop

The loop indicates that information and learnings from the production environment are fed back into the earlier stages of the pipeline. This feedback can be used to improve the software development process and future releases.

How DevOps Works?

- DevOps will remove the Isolation conditions between the development team and operations team.
- In many cases these two teams will work together for the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

- Teams in charge of security and quality assurance may also integrate more closely with development and operations over the course of an application's lifecycle under various DevOps models.
- DevSecOps is the term used when security is a top priority for all members of a DevOps team.

DevOps Life Cycle

- DevOps is a practice that enables a single team to handle the whole application lifecycle, including development, testing, release, deployment, operation, display, and planning. It is a mix of the terms “Dev” (for development) and “Ops” (for operations).
- We can speed up the delivery of applications and services by a business with the aid of DevOps.
- Amazon, Netflix, and other businesses have all effectively embraced DevOps to improve their customer experience.

- **DevOps Lifecycle** is the set of phases that includes DevOps for taking part in Development and Operation group duties for quicker software program delivery.
- DevOps follows positive techniques that consist of **code, building, testing, releasing, deploying, operating, displaying, and planning.**
- **DevOps lifecycle** follows a range of phases such as non-stop development, non-stop integration, non-stop testing, non-stop monitoring, and non-stop feedback.

innovate achieve lead

7 Cs of DevOps

- Continuous Development
- Continuous Integration
- Continuous Testing
- Continuous Deployment/Continuous Delivery
- Continuous Monitoring
- Continuous Feedback
- Continuous Operations

Benefits of DevOps

Faster Delivery: DevOps enables organizations to release new products and updates faster and more frequently, which can lead to a competitive advantage.

Improved Collaboration: DevOps promotes collaboration between development and operations teams, resulting in better communication, increased efficiency, and reduced friction.

Improved Quality: DevOps emphasizes automated testing and continuous integration, which helps to catch bugs early in the development process and improve the overall quality of software.

Increased Automation: DevOps enables organizations to automate many manual processes, freeing up time for more strategic work and reducing the risk of human error.

Better Scalability: DevOps enables organizations to quickly and efficiently scale their infrastructure to meet changing demands, improving the ability to respond to business needs.

Increased Customer Satisfaction: DevOps helps organizations to deliver new features and updates more quickly, which can result in increased customer satisfaction and loyalty.

Improved Security: DevOps promotes security best practices, such as continuous testing and monitoring, which can help to reduce the risk of security breaches and improve the overall security of an organization's systems.

Better Resource Utilization: DevOps enables organizations to optimize their use of resources, including hardware, software, and personnel, which can result in cost savings and improved efficiency

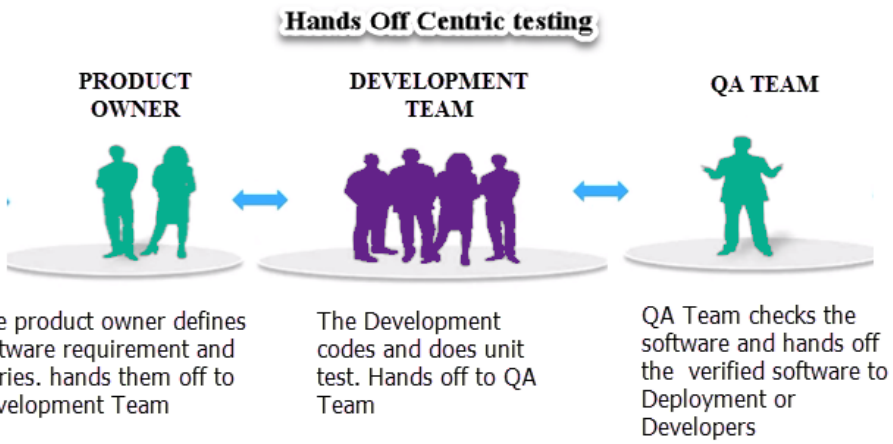
Continuous Testing in DevOps

innovate achieve lead

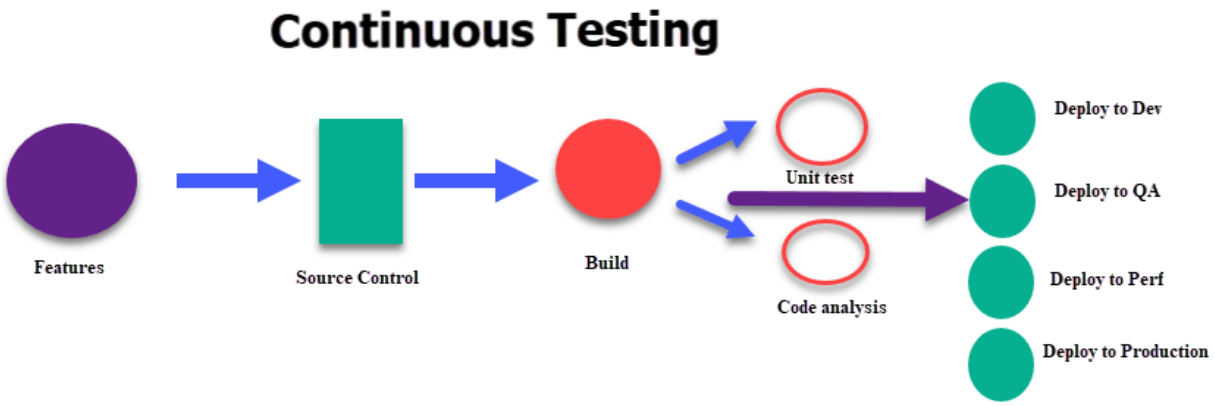
- Fixing a software bug after a product is launched can be up to 100 times more expensive than addressing it during the development phase.
- Continuous Testing is a vital approach in modern software development, where testing is integrated at every stage of the process.
- This helps identify errors early, reducing both the risk of defects and the costs that come with fixing them later.

- **Continuous Testing** refers to running automated tests every time code changes are made, providing fast feedback as part of the software delivery pipeline.
- It was introduced to help developers quickly identify, notify, and fix issues. The goal is to test more frequently, starting with individual components and later testing the entire codebase.
- Continuous Testing is a key part of the Continuous Integration (CI) process within Agile and DevOps pipelines, enabling faster and more efficient software delivery.

How is Continuous Testing different?



- Continuous means undisrupted testing done on a continuous basis.
- In a Continuous DevOps process, a software change (release candidate) is continuously moving from Development to Testing to Deployment.
- The code is continuously developed, delivered, tested and deployed.



For Example,

- whenever a developer checks the code in the Source Code Server like Jenkins automated set of unit tests are executed in the continuous process.
- If the tests fail, the build is rejected, and the developer is notified. If the build passes the test, it is deployed to performance, QA servers for exhaustive functional and load tests.
- The tests are run in parallel. If the tests pass, the software is deployed in production.

How Is Continuous Testing Different from Test Automation?

Parameter	Test Automation	Continuous Testing
Definition	Test automation is a process where tool or software is used for automating tasks.	It is a software testing methodology which focuses on achieving continuous quality & improvement.
Purpose	A set of similar or repetitive tasks, a machine can execute, faster, with a fewer mistake.	The continuous testing process helps to find the risk, address them and improve the quality of the product.
Prerequisite	Automation in testing possible without integrating continuous testing.	Continuous testing can not be implemented without test automation.

Time	Software release can take a month to years.	Software release may be released weekly to hourly.
Feedback	Regular feedback after testing each release.	Feedback at each stage needs to be instant.
History	Automated testing has been done for decades to make the testing process faster.	Continuous testing is a relatively newer concept.

Continuous Testing Tools

1) QuerySurge

QuerySurge is the smart data testing solution that is the first-of-its-kind full DevOps solution for continuous data testing. Key features include Robust API with 60+ calls, detailed data intelligence & data analytics, seamless integration into the DevOps pipeline for continuous testing, and verifies large amounts of data quickly.

2) Jenkins

Jenkins is a Continuous Integration tool which is written using Java language. This tool can be configured via GUI interface or console commands.

3) Travis

Travis is continuous testing tool hosted on the GitHub. It offers hosted and on-premises variants. It provides a variety of different languages and a good documentation.

4) Selenium

Selenium is open-source software testing tool. It supports all the leading browsers like Firefox, Chrome, IE, and Safari. Selenium WebDriver is used to automate web application testing.

innovate achieve lead

THANK YOU