BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Session 9 – Revision : EC2 Mid Semester

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.

SOFTWARE ERRORS, DEFECTS, AND FAILURES

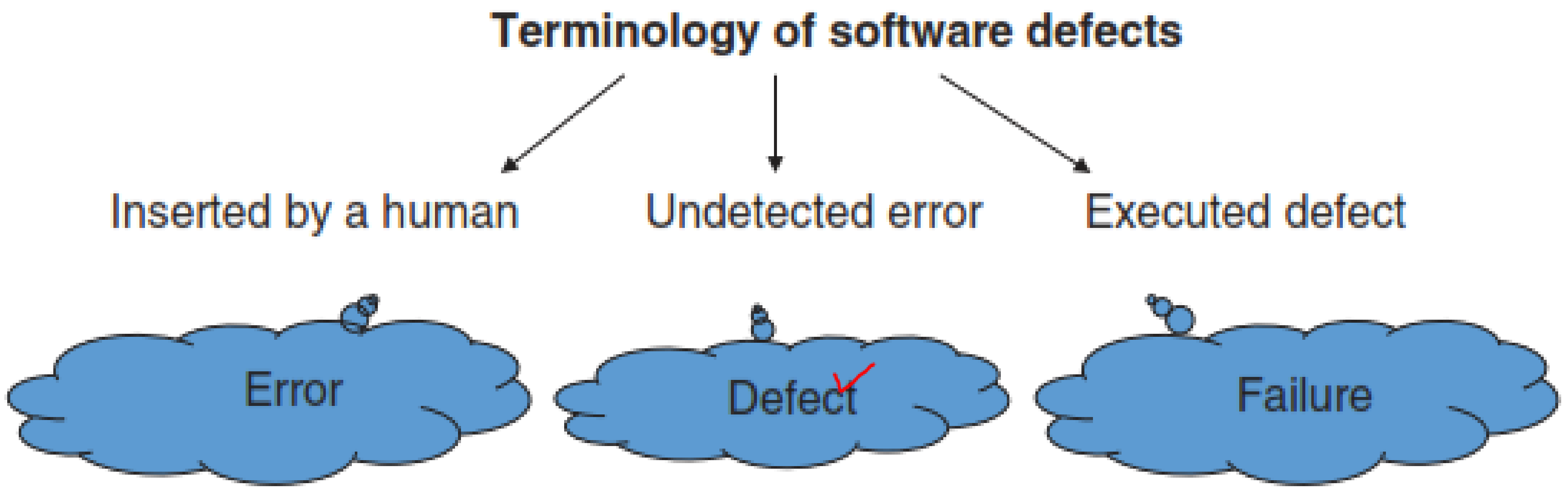


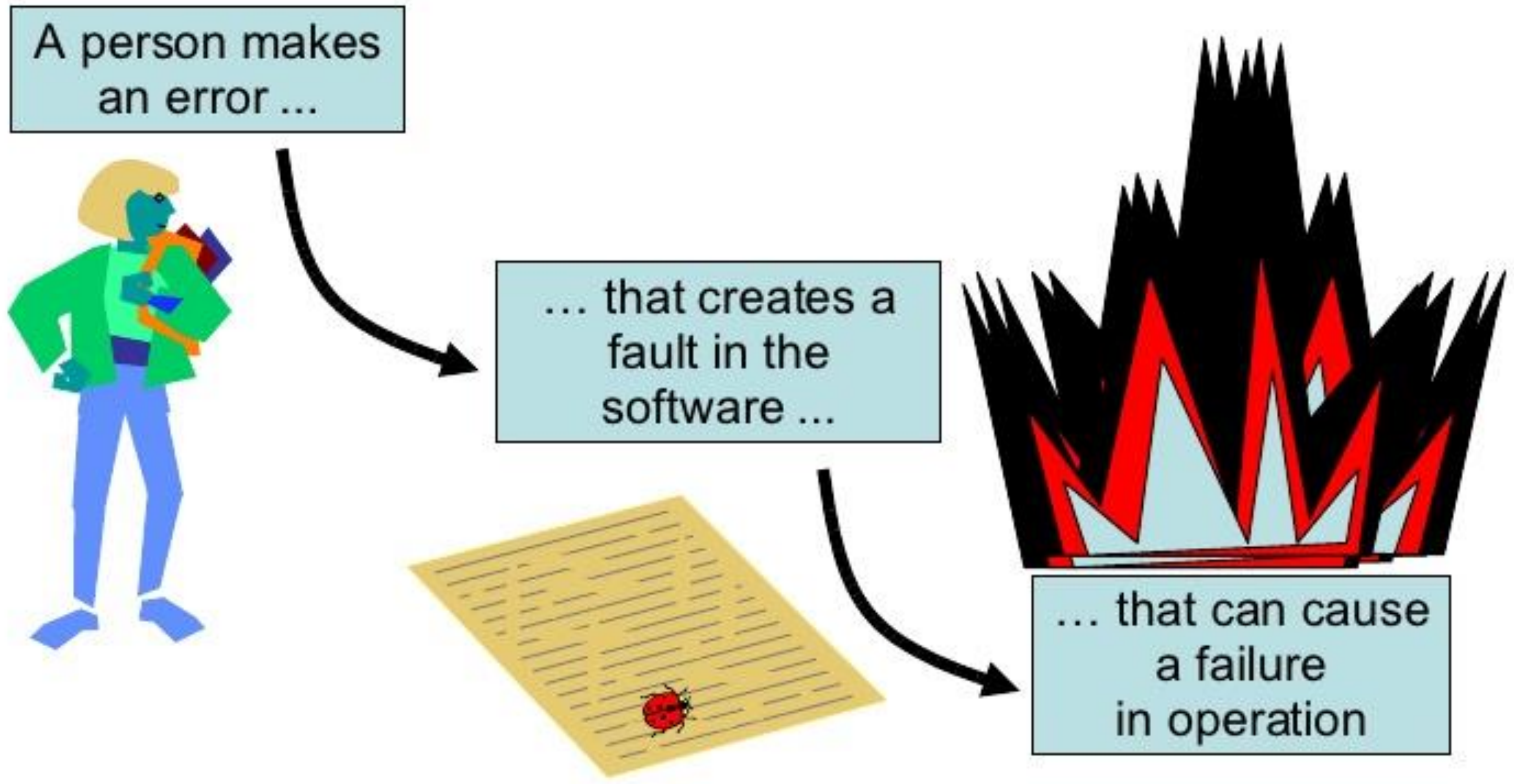
Figure 1.2 Terminology recommended for describing software problems.

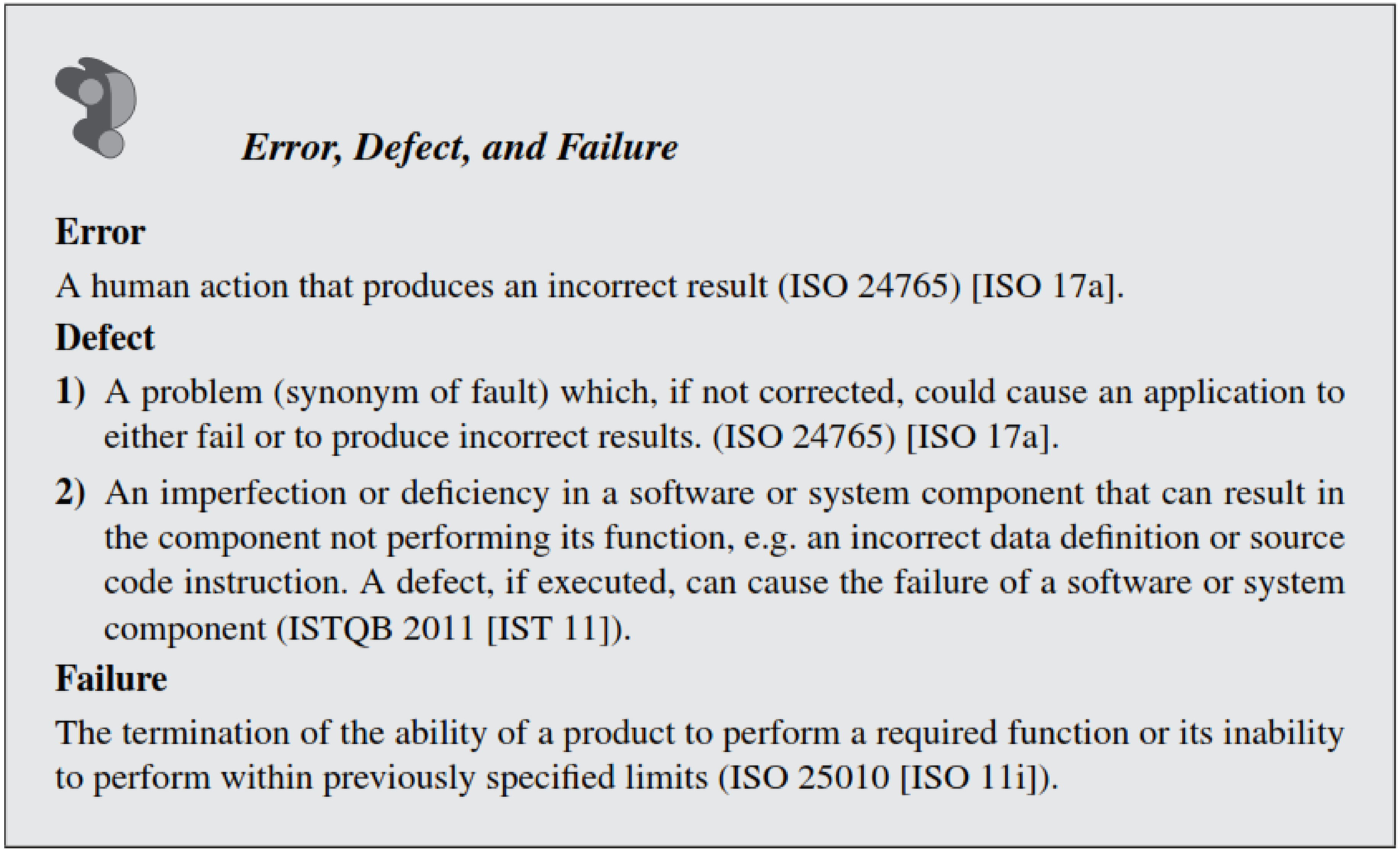
- A failure (synonymous with a crash or breakdown) is the execution (or manifestation) of a fault in the operating environment.
- A failure is defined as the **termination of the ability of a component to fully or partially perform a function** that it was designed to carry out.
- The origin of a failure 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 - Fault - Failure


A person makes an error ...
… that creates a fault in the software ...
… that can cause a failure in operation



Error, Defect, and Failure

Error

A human action that produces an incorrect result (ISO 24765) [ISO 17a].

Defect

1) 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].
2) An imperfection or deficiency in a software or system component that can result in the component not performing its function, e.g. 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 11]).

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]).

Software Quality Issues in an Online Event Ticket Booking System

A small **event ticket booking website** is developed to allow customers to **select seats, add them to their booking cart, and proceed to checkout**. However, some issues arise during testing and after deployment:

1. **A developer accidentally writes incorrect code for seat selection**. Instead of reserving the selected seat, the system removes it, making it unavailable for the user.
2. **During testing, a tester tries to book two seats**, but instead of confirming the booking, the system shows **an empty cart**, meaning the seats were not added properly.
3. **After launch, many customers complain that their selected seats disappear** after adding them to the cart, making them unable to proceed with the payment.

Questions

i) Define and differentiate between defect, error, and failure using the given case.

ii) What methods can be used to identify each of these issues, and what strategies should be employed to fix them?

i) Explanation of Defect, Error, and Failure in the Given Case Study

In the given scenario, the terms **error, defect, and failure** represent different stages of software issues:

1. **Error (Mistake by Developer)**
   - An **error** occurs when a developer writes incorrect code due to a logical or syntactical mistake.
   - In this case, the **developer mistakenly writes incorrect code for seat selection**, causing the system to remove the seat instead of reserving it.
2. **Defect (Bug Found During Testing)**
   - A **defect** (or bug) is an issue found during testing when an error in the code results in incorrect functionality.
   - Here, during testing, the **tester tries to book two seats, but the cart appears empty**, indicating a flaw in the booking logic.
3. **Failure (Issue in Production Affecting Users)**
   - A **failure** occurs when a defect is not fixed before deployment and causes real-world issues for users.
   - After launch, **many customers complain that their selected seats disappear**, preventing them from completing their bookings. This represents a failure because it impacts the end-user experience.

ii) Methods to Identify and Fix These Issues

Issue Type	Methods to Identify	Strategies to Fix
Error (Developer Mistake)	- **Code reviews** to detect incorrect logic. - **Static code analysis** tools to check for coding errors. - **Unit testing** for seat reservation functions.	- Implement **peer code reviews** before merging changes. - Follow **Test-Driven Development (TDD)** to validate functionality. - Use **debugging tools** to verify seat reservation logic.
Defect (Bug Found in Testing)	- **Functional testing** to ensure seats are added correctly. - **UI testing** to verify cart updates. - **Regression testing** after fixing bugs to check for unintended side effects.	- Fix the **cart update logic** in the code. - Improve **test cases** to include seat selection and addition. - Automate testing to catch such issues earlier.
Failure (Customer Complaints in Production)	- **User feedback & support tickets** to track complaints. - **Real-time monitoring & logging** to analyze why seats disappear. - **Automated UI tests** to simulate real customer actions.	- Deploy **hotfixes** to resolve the issue in production. - Improve **error logging & alert systems** to catch failures early. - Implement a **staging environment** for testing before live deployment.

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"

Quality assurance

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.

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.

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.

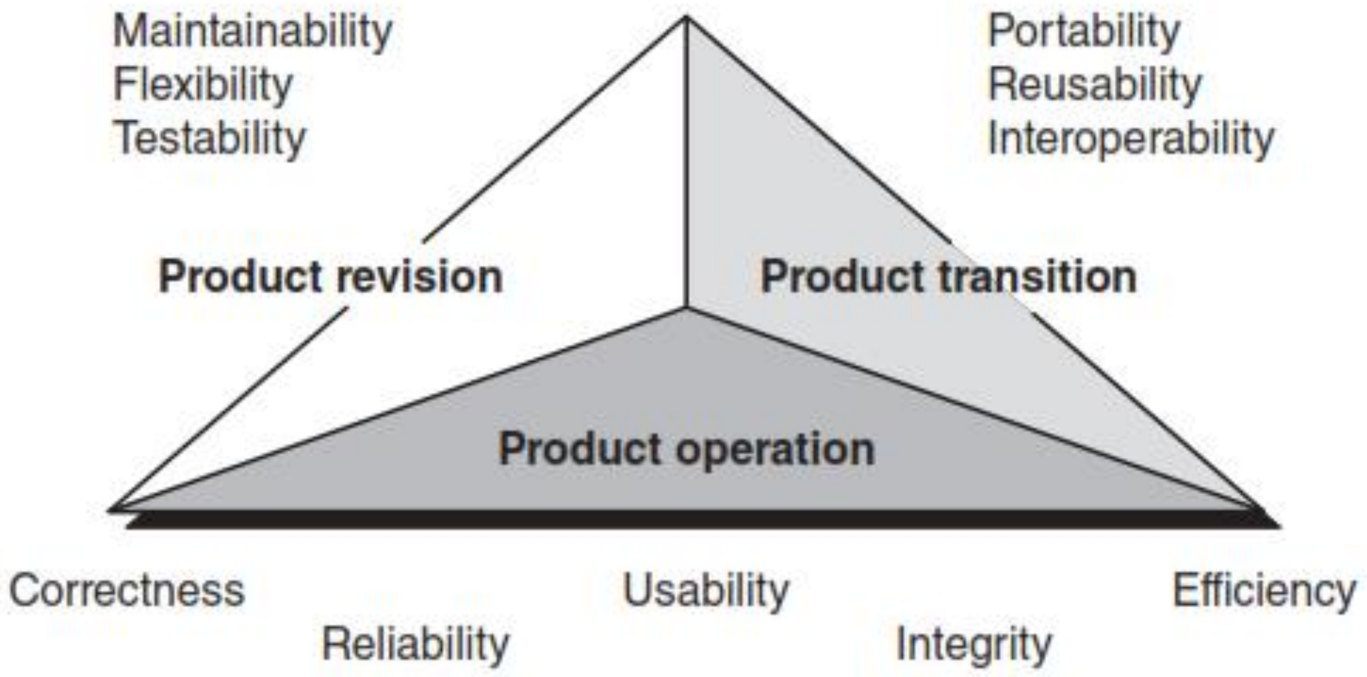


Figure 3.1 The three perspectives and 11 quality factors of McCall et al. (1977) [MCC 77].

- 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).

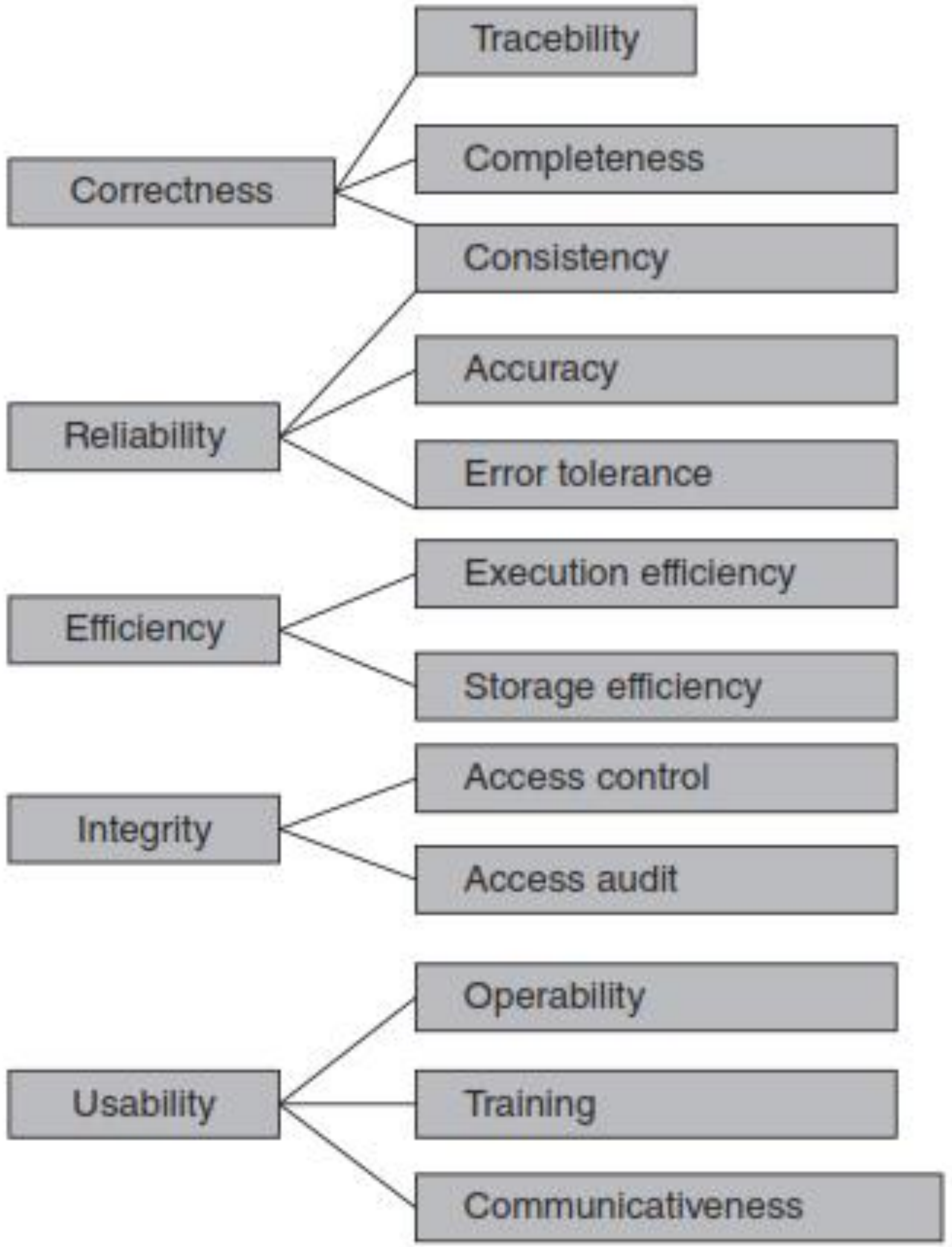


Figure 3.2 Quality factors and criteria from McCall et al. (1977) [MCC 77].

Scenario: Balancing Speed and Safety: The Software Quality Assurance Dilemma in Autonomous Vehicle Development

A technology company is developing an **autonomous vehicle software** that enables **self-driving cars to navigate, detect obstacles, and ensure passenger safety**. Management is debating whether to **invest in extensive Software Quality Assurance (SQA) early** or focus on **rapid development and deployment to gain a competitive edge**.

Some team members argue that **early investment in software testing and safety validation** ensures reliability and prevents critical failures, while others believe that **faster deployment** will help establish market dominance. However, **neglecting quality early** could result in **accidents, software malfunctions, and legal liabilities**, ultimately harming the company's reputation and customer trust.

1. **Why is it crucial and financially beneficial to prioritize Software Quality Assurance (SQA) in autonomous vehicle systems?** Discuss the impact of **software failures, safety concerns, regulatory requirements, and long-term system reliability** on the success and sustainability of self-driving technology.

2. **How does reducing prevention costs impact the total cost of software quality?** Explain the connection between **prevention, appraisal, and failure costs** in software development.

Answers

1. Importance of Prioritizing Software Quality Assurance (SQA) in Autonomous Vehicle Systems

- **Cost of Failures:** Software defects in autonomous vehicles can lead to **accidents, lawsuits, and expensive recalls.** Fixing issues after deployment is far more costly than preventing them early.
- **Safety Risks:** Bugs in self-driving software can cause **incorrect navigation, obstacle misdetection, or braking failures**, risking lives.
- **Regulatory Compliance:** Meeting **safety and legal requirements** (e.g., ISO 26262) is essential to avoid **fines and operational bans.**
- **Long-Term Reliability:** High-quality software **builds trust, reduces maintenance costs, and ensures system stability**, leading to long-term business success.

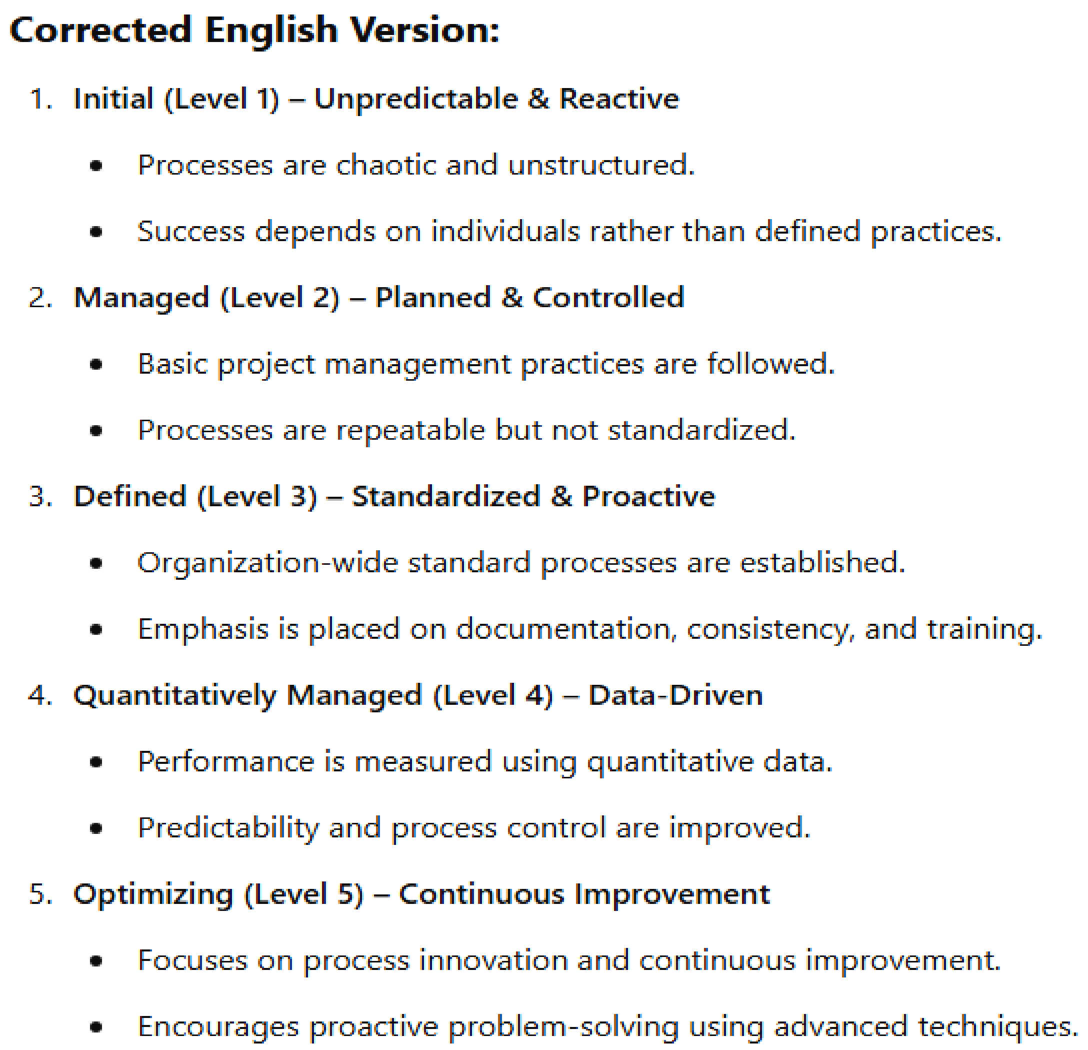
2. Impact of Reducing Prevention Costs on the Total Cost of Software Quality

In software development, **reducing prevention costs** (such as early testing, code reviews, and security checks) can increase overall quality costs due to expensive failures later in the development cycle.

2.1 Relationship Between Prevention, Appraisal, and Failure Costs

Cost Type	Description	Impact of Reducing Costs
Prevention Costs	Costs related to **early quality measures**, such as testing, training, and code reviews.	Reducing prevention efforts increases the chance of defects appearing later.
Appraisal Costs	Costs related to **inspecting and testing software** before release to find defects.	Less investment in testing can allow defects to go unnoticed.
Failure Costs	Costs due to **bugs and failures** in production, including fixes, customer compensation, and legal fines.	Higher failure costs due to **post-release defects, recalls, or security breaches.**

CMMI-DEV (Capability Maturity Model Integration for Development)



Corrected English Version:

1. **Initial (Level 1) – Unpredictable & Reactive**
   - Processes are chaotic and unstructured.
   - Success depends on individuals rather than defined practices.
2. **Managed (Level 2) – Planned & Controlled**
   - Basic project management practices are followed.
   - Processes are repeatable but not standardized.
3. **Defined (Level 3) – Standardized & Proactive**
   - Organization-wide standard processes are established.
   - Emphasis is placed on documentation, consistency, and training.
4. **Quantitatively Managed (Level 4) – Data-Driven**
   - Performance is measured using quantitative data.
   - Predictability and process control are improved.
5. **Optimizing (Level 5) – Continuous Improvement**
   - Focuses on process innovation and continuous improvement.
   - Encourages proactive problem-solving using advanced techniques.

Refer Session 3 PPT for more detail

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.**

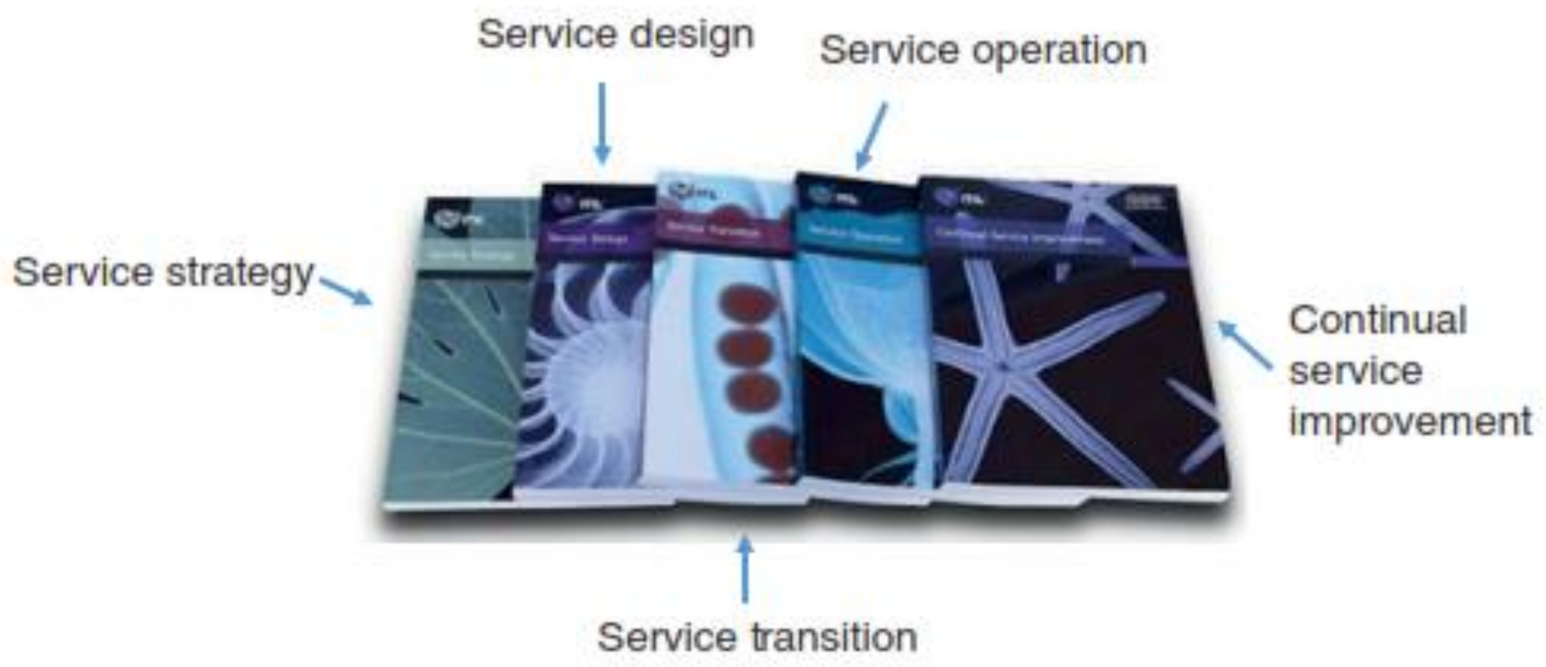
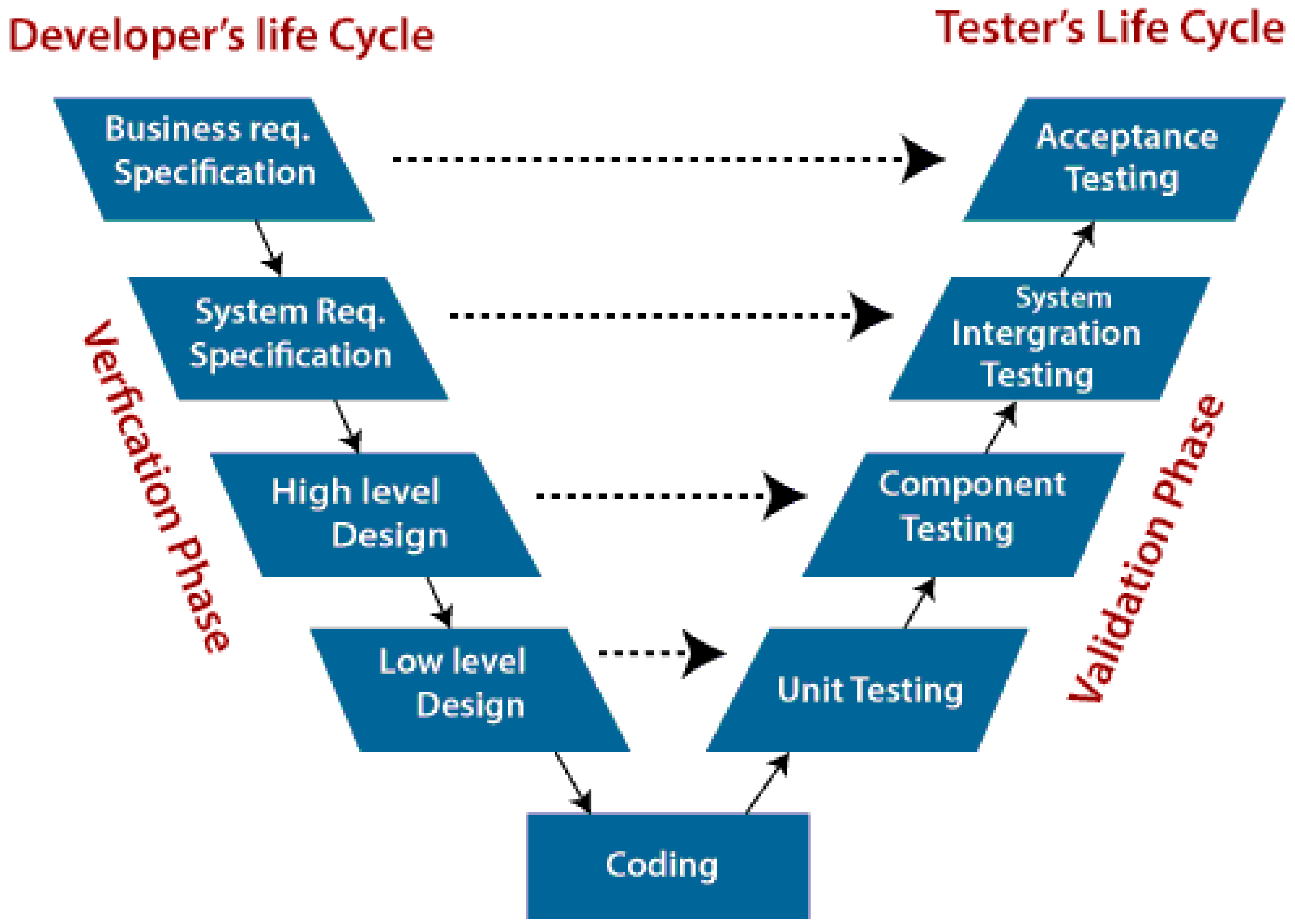


Figure 4.11 The main ITIL guides.

Software Testing

Dynamic Testing: This begins when the code or a specific unit/module is ready. It involves executing the code to validate its functionality. Common techniques for dynamic testing include black-box testing (focusing on inputs and outputs without knowing internal details), gray-box testing (a mix of internal knowledge and external testing), and white-box testing (testing internal structures and logic)..

V- Model


Developer's life Cycle
Tester's Life Cycle
Business req. Specification
System Req. Specification
High level Design
Low level Design
Coding
Unit Testing
Component Testing
System Intergration Testing
Acceptance Testing
Verfication Phase
Validation Phase

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.

BOUNDARY VALUE ANALYSIS (BVA)

It is a **black-box testing technique** based on the principle that defects are more likely to occur at the boundaries of input ranges rather than in the middle. This is because boundary conditions are more prone to errors due to incorrect implementation of logical conditions. 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.

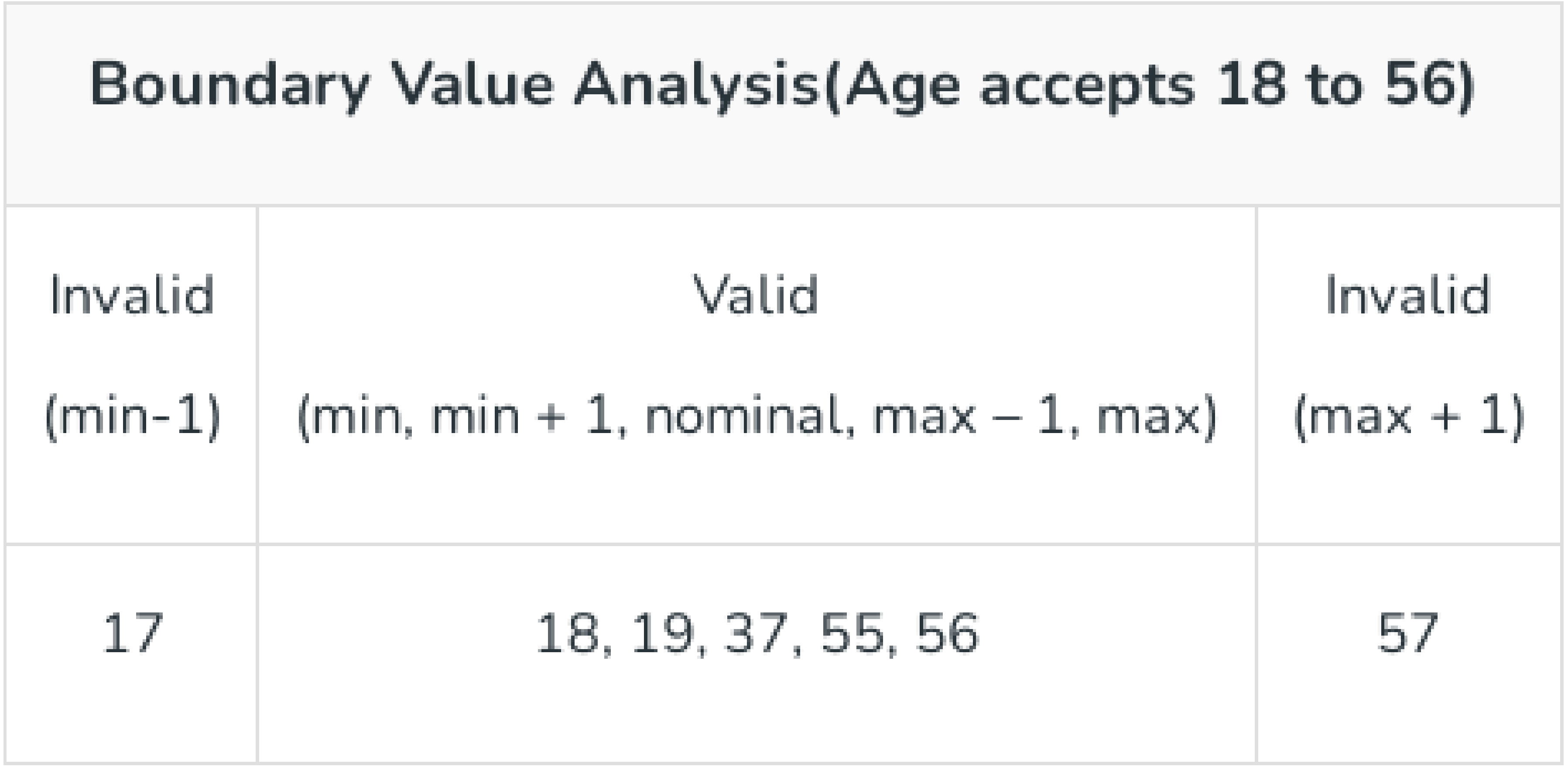
iii. The requirements themselves may not be clearly understood, especially around the boundaries, thus causing even the correctly coded program to not perform the correct way.

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}

- When more than one variable for the same application is checked then one can use a single fault assumption.
- Holding all variables at their **minimum or maximum values** while changing only **one variable at a time** to its extreme value for testing.

Consider a system that accepts ages from 18 to 56.



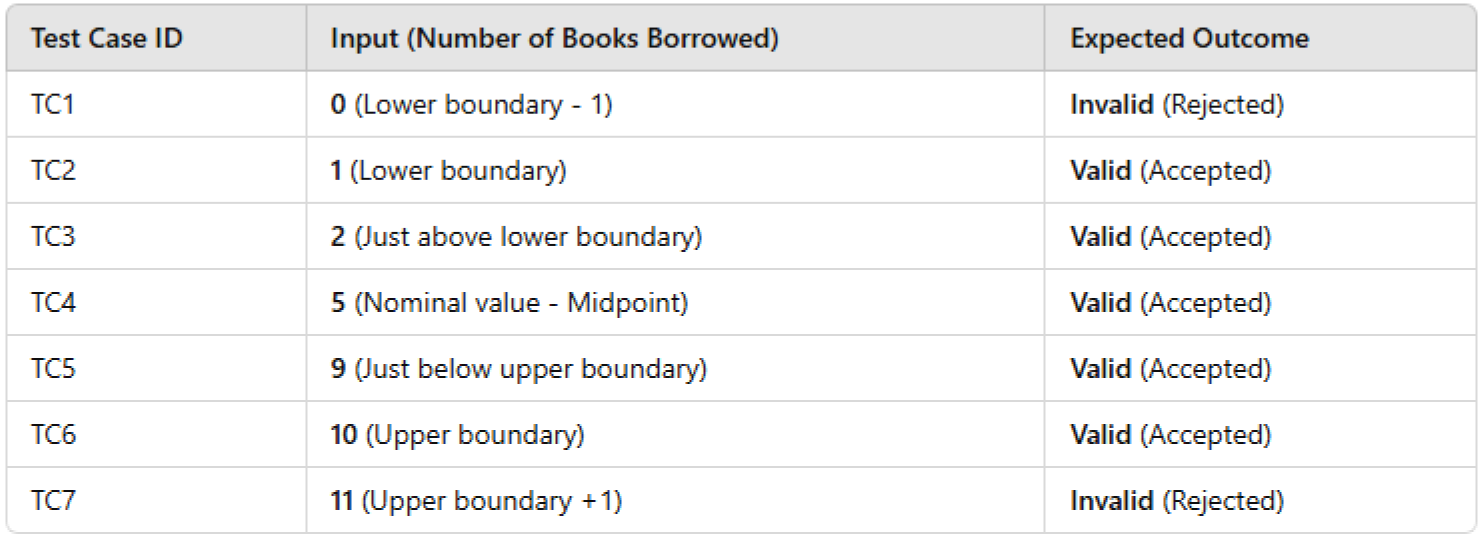
Boundary Value Analysis(Age accepts 18 to 56)		
Invalid (min-1)	Valid (min, min + 1, nominal, max – 1, max)	Invalid (max + 1)
17	18, 19, 37, 55, 56	57

Boundary Value Analysis (BVA) for Library Book Borrowing Limits

library management system is undergoing testing to verify that it correctly processes and validates the number of books a user can borrow. The system permits users to borrow between **1 and 10 books**.

Using **Boundary Value Analysis (BVA)**, identify the appropriate test cases to ensure that the system accurately handles both valid and invalid inputs. List the test cases along with their expected results.

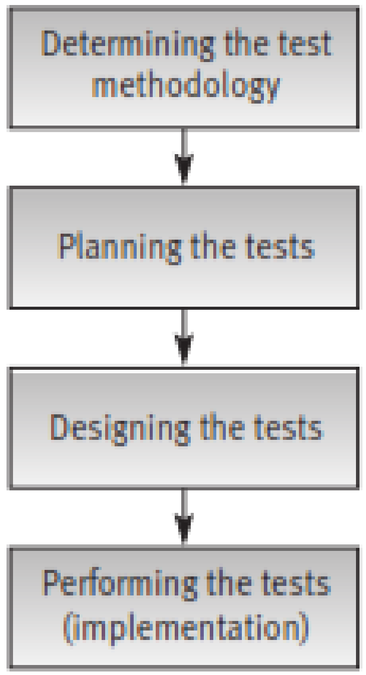
Answer



Test Case ID	Input (Number of Books Borrowed)	Expected Outcome
TC1	**0** (Lower boundary - 1)	**Invalid** (Rejected)
TC2	**1** (Lower boundary)	**Valid** (Accepted)
TC3	**2** (Just above lower boundary)	**Valid** (Accepted)
TC4	**5** (Nominal value - Midpoint)	**Valid** (Accepted)
TC5	**9** (Just below upper boundary)	**Valid** (Accepted)
TC6	**10** (Upper boundary)	**Valid** (Accepted)
TC7	**11** (Upper boundary +1)	**Invalid** (Rejected)

The testing process

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


Determining the test methodology
Planning the tests
Designing the tests
Performing the tests (implementation)

The testing process

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.

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 performance of these tests, the test cases are recorded into 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.

*Load tests :*The history of software system development contains many sad chapters of systems that succeeded in correctness tests but severely failed – and caused enormous damage – once they were required to operate under standard full load.

The damage in many cases was extremely high because the failure occurred “unexpectedly”, when the systems were supposed to start providing their regular software services.

*Test management :*Testing involves **many participants occupied in actually carrying out the tests and correcting the detected errors**.

Testing typically **monitors 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.