



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan





SE ZG501

Software Quality Assurance and

Testing

Lecture No. 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



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.**

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**.

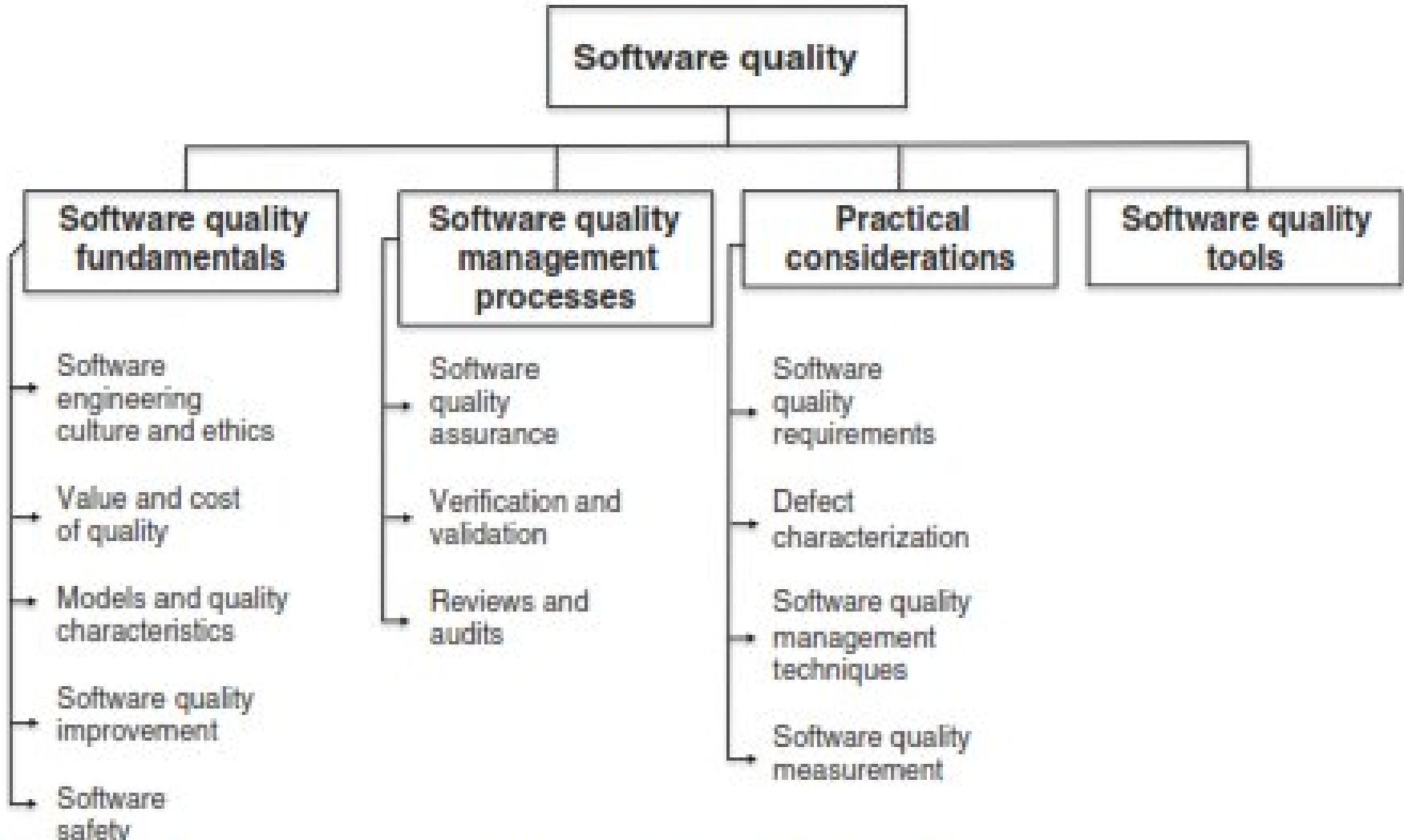


Figure 1.1 Software Quality in the SWEBOK® Guide [SWE 14].

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.

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.

Terminology of software defects

Inserted by a human

Undetected error

Executed defect

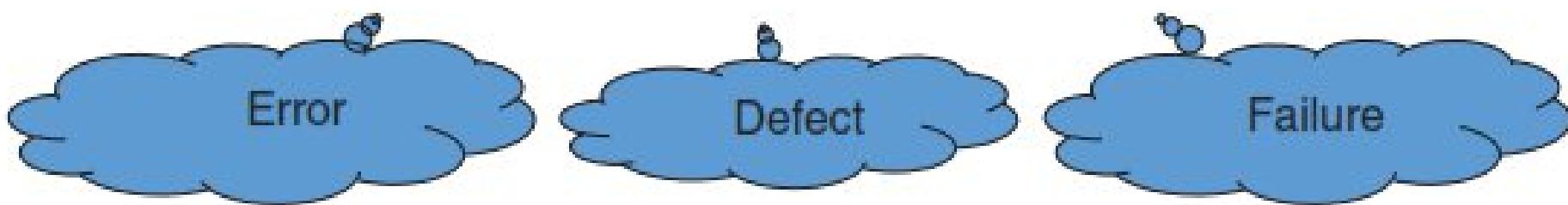


Figure 1.2 Terminology recommended for describing software problems.

17/06

9/9

0800 Andon started
 1000 stopped - andon ✓
 1200 (020) MP - nc
 020 PRO ✓
 const 2.13042676
 const 2.13067646

Relay 6-2 in 021 field ground open test
 in relay - open test.

1100 Started cosine Taps (Sine chart)
 1525 Started Multi Adder Test.

1505



Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.
 After debug start.
 1700 closed form.

A **failure** (synonymous with a crash or breakdown) is the execution (or manifestation) of a fault in the operating environment.

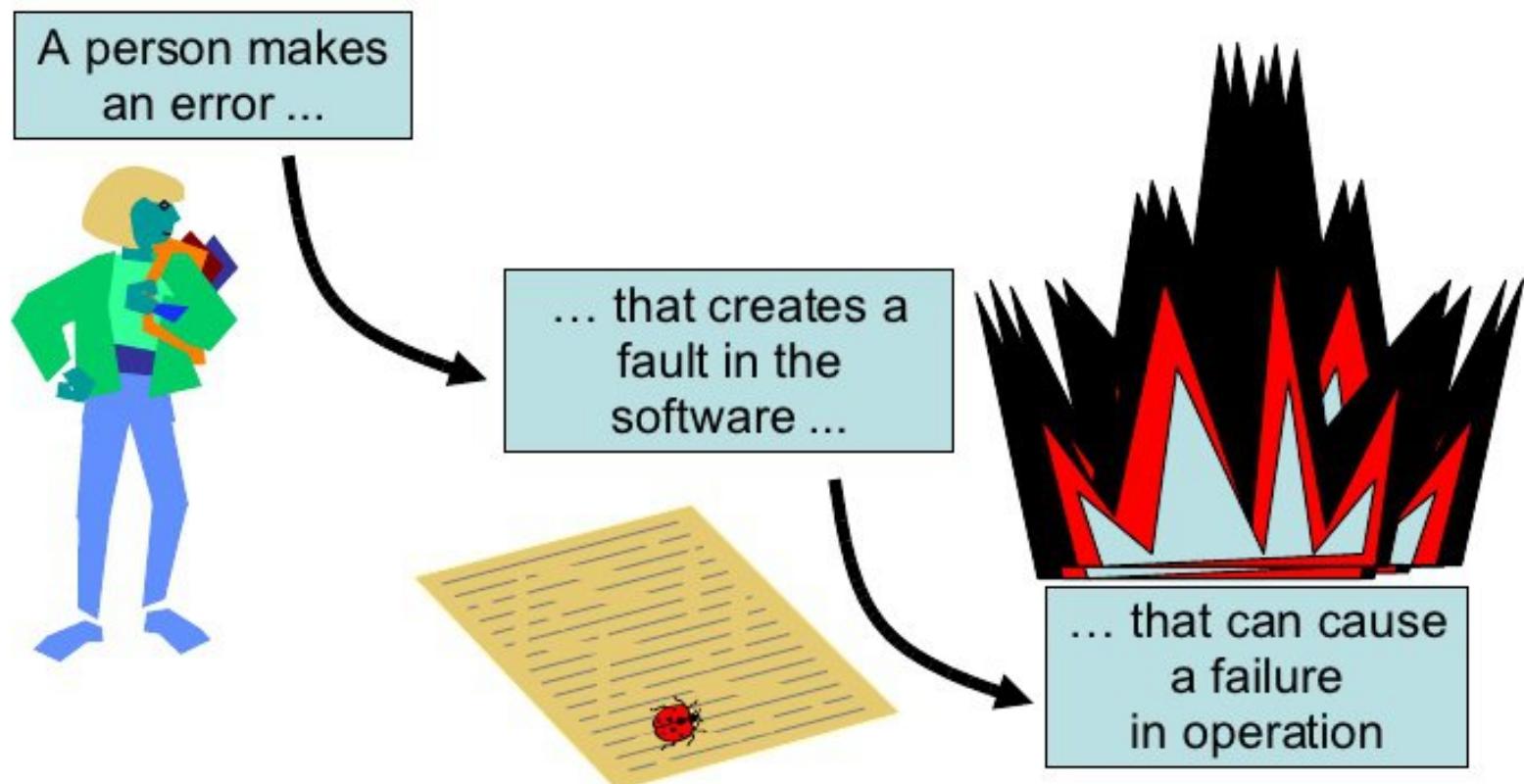
A failure is defined as the **termination of the ability of a component to fully or partially perform a function** that it was designed to carry out.

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





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]).

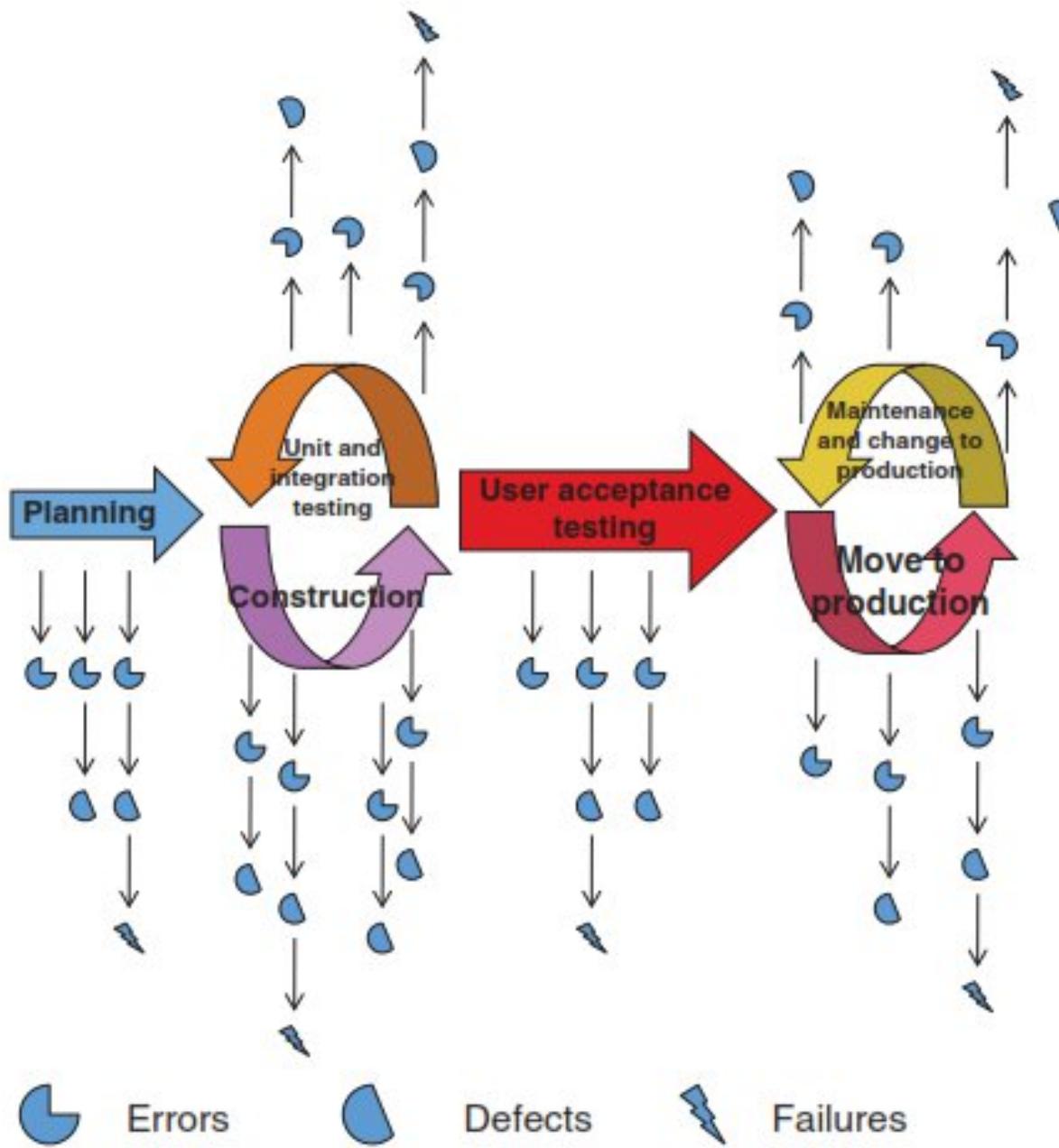


Figure 1.3 Errors, defects, and failures in the software life cycle.

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.

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.

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!**

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.

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.**

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*

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 confidence 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**.

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**.

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 instil 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**.

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.



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.
-

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.



The objectives of SQA activities

Software development (process-oriented):

Software maintenance (product-oriented):

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.**

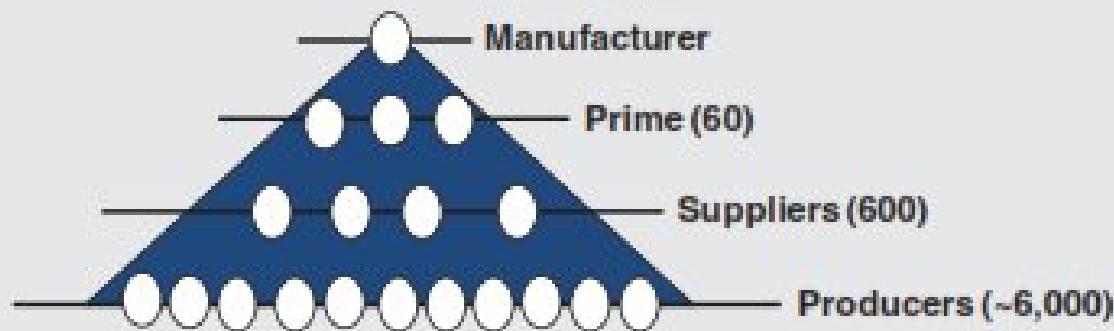
Software maintenance (product-oriented):

- 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.

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.

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**.

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.

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 error¹ [JON 00].

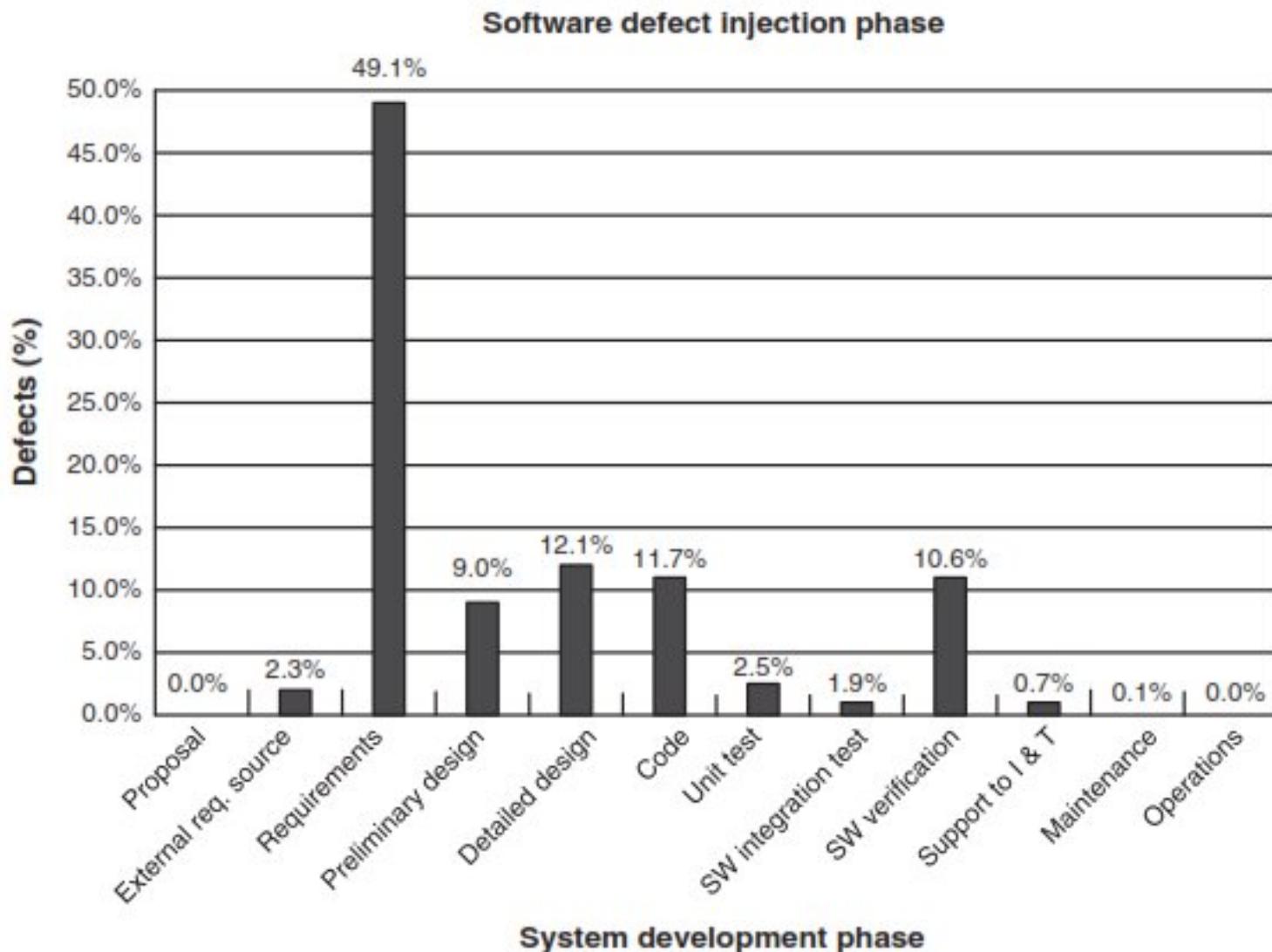


Figure 2.3 Defect injection distribution during the life cycle [SEL 07].



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan





SE ZG501

Software Quality Assurance and

Testing

Lecture No. 2

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**

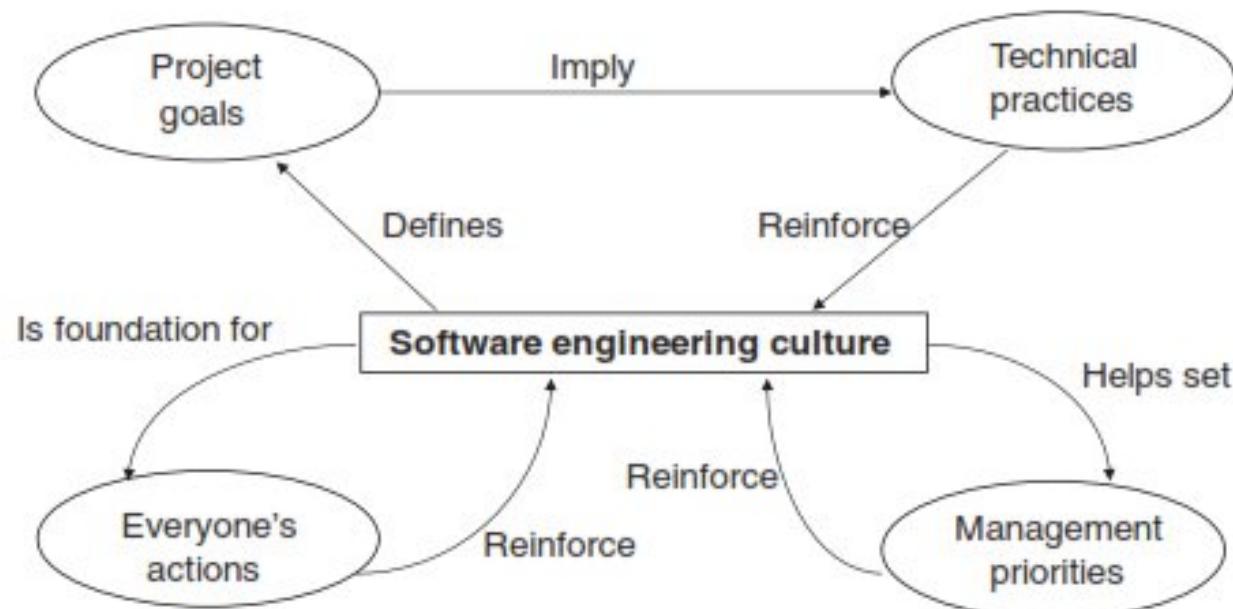


Figure 2.5 Software engineering culture.

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.

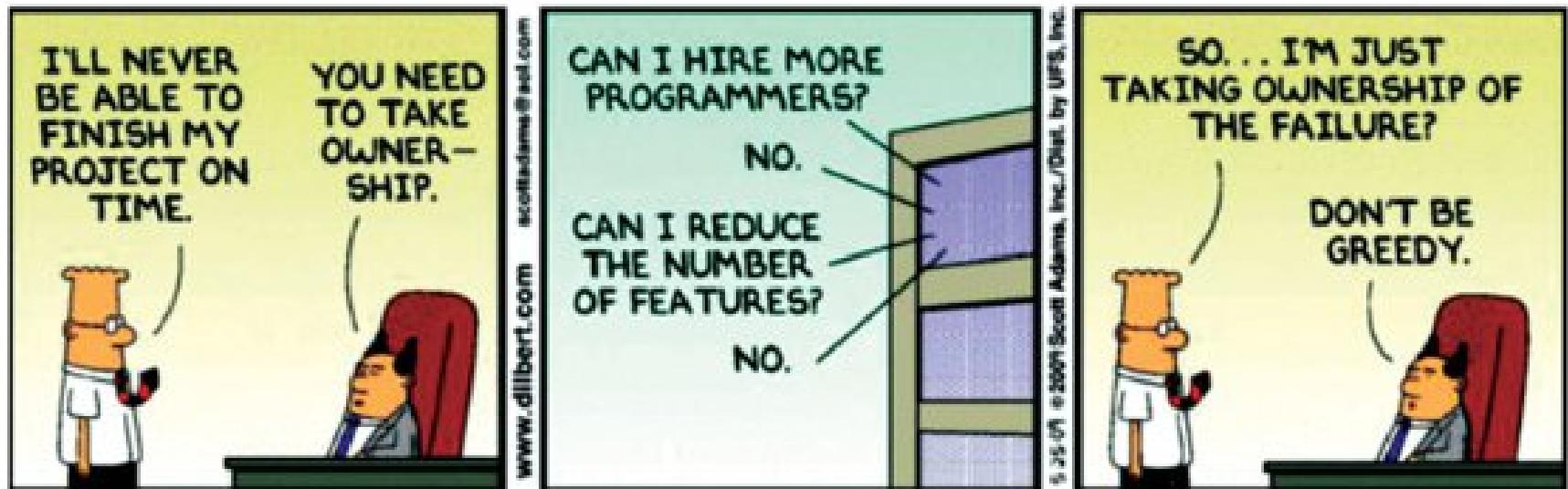


Figure 2.8 Dilbert tries to negotiate a change in his project. DILBERT © 2009 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

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.

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.

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.

The role QA plays in the implementation stage:

- Code Reviews
- System Integration Testing
- User Acceptance Testing

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.

The Role of QA in Deployment

In the deployment stage, QA ensures that *all elements of the custom software development process are properly implemented, tested, verified, and deployed*. This ensures the product is released with confidence, free from issues or bugs.

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!

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)

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 1220TM-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]

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.

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.

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.

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.

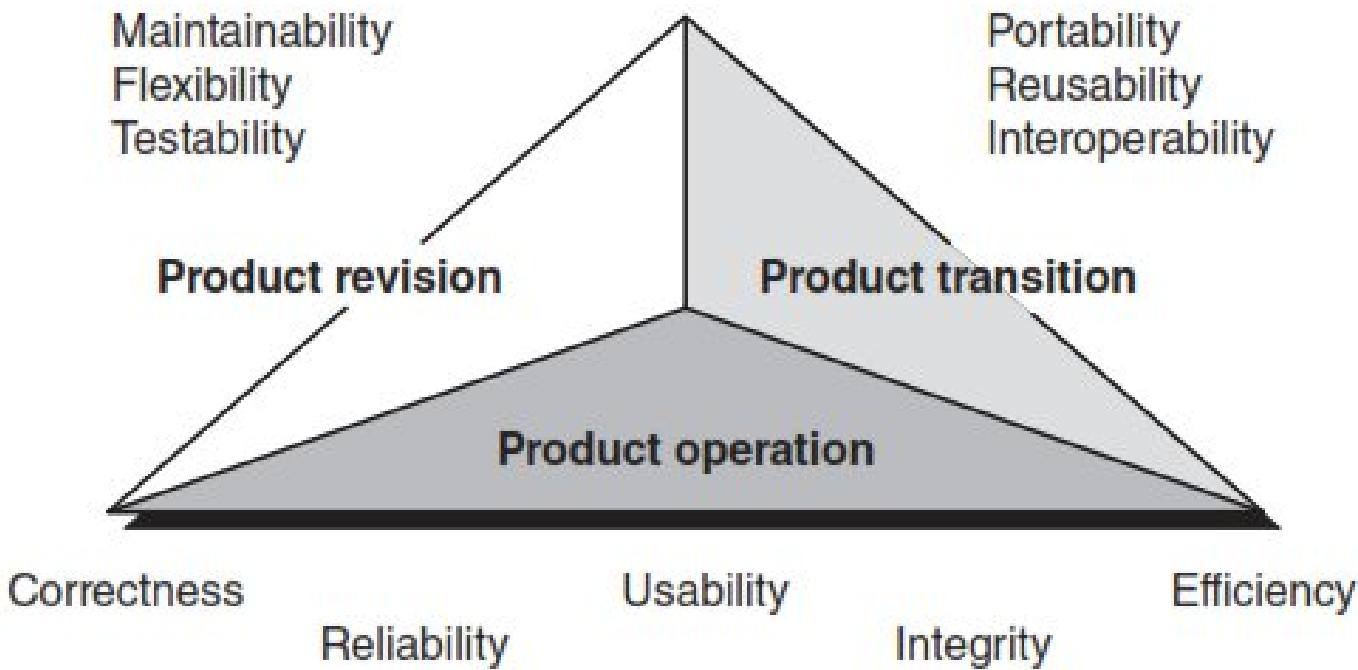


Figure 3.1 The three perspectives and 11 quality factors of McCall et al. (1977) [MCC 77].

- 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).

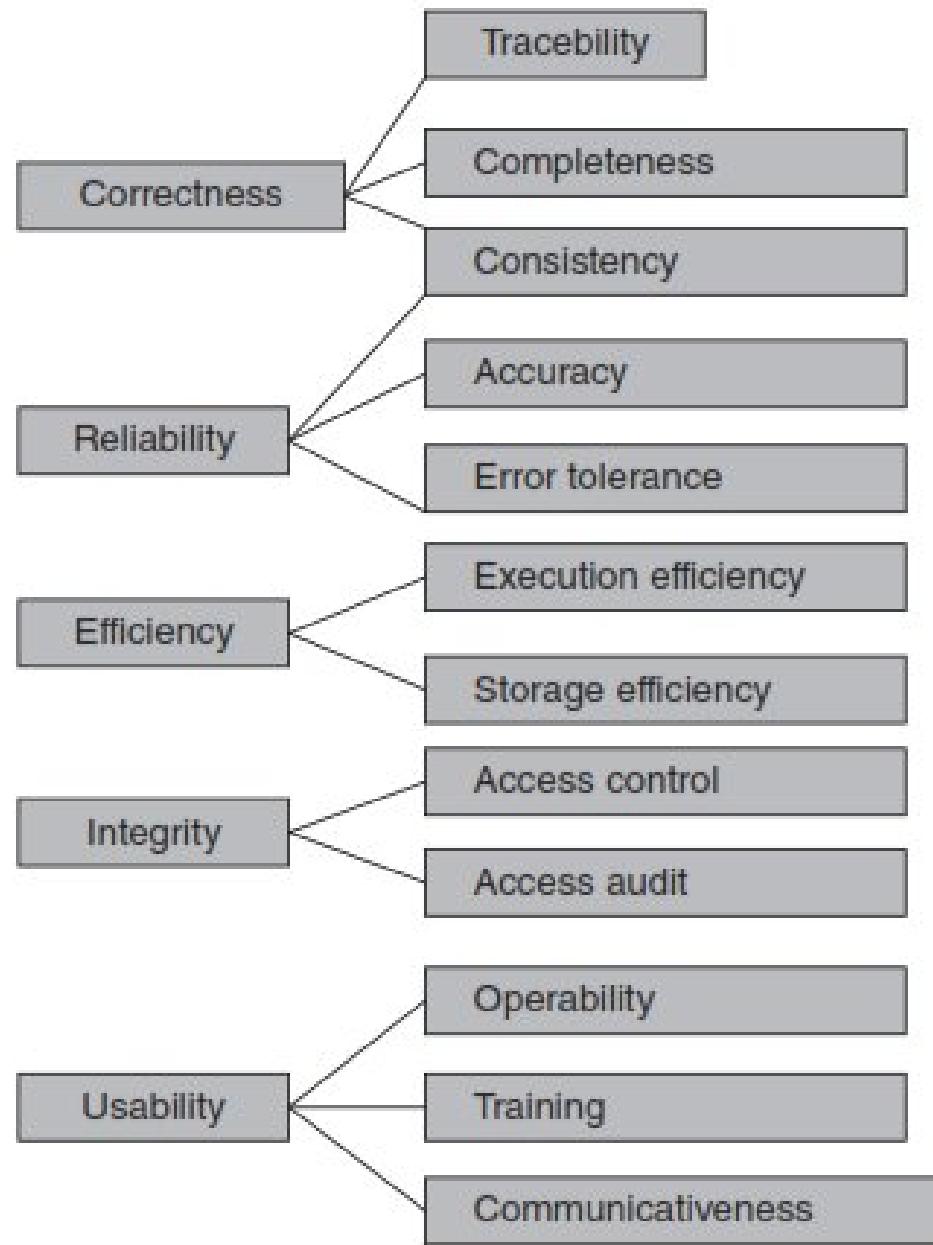


Figure 3.2 Quality factors and criteria from McCall et al. (1977) [MCC 77].

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.

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.

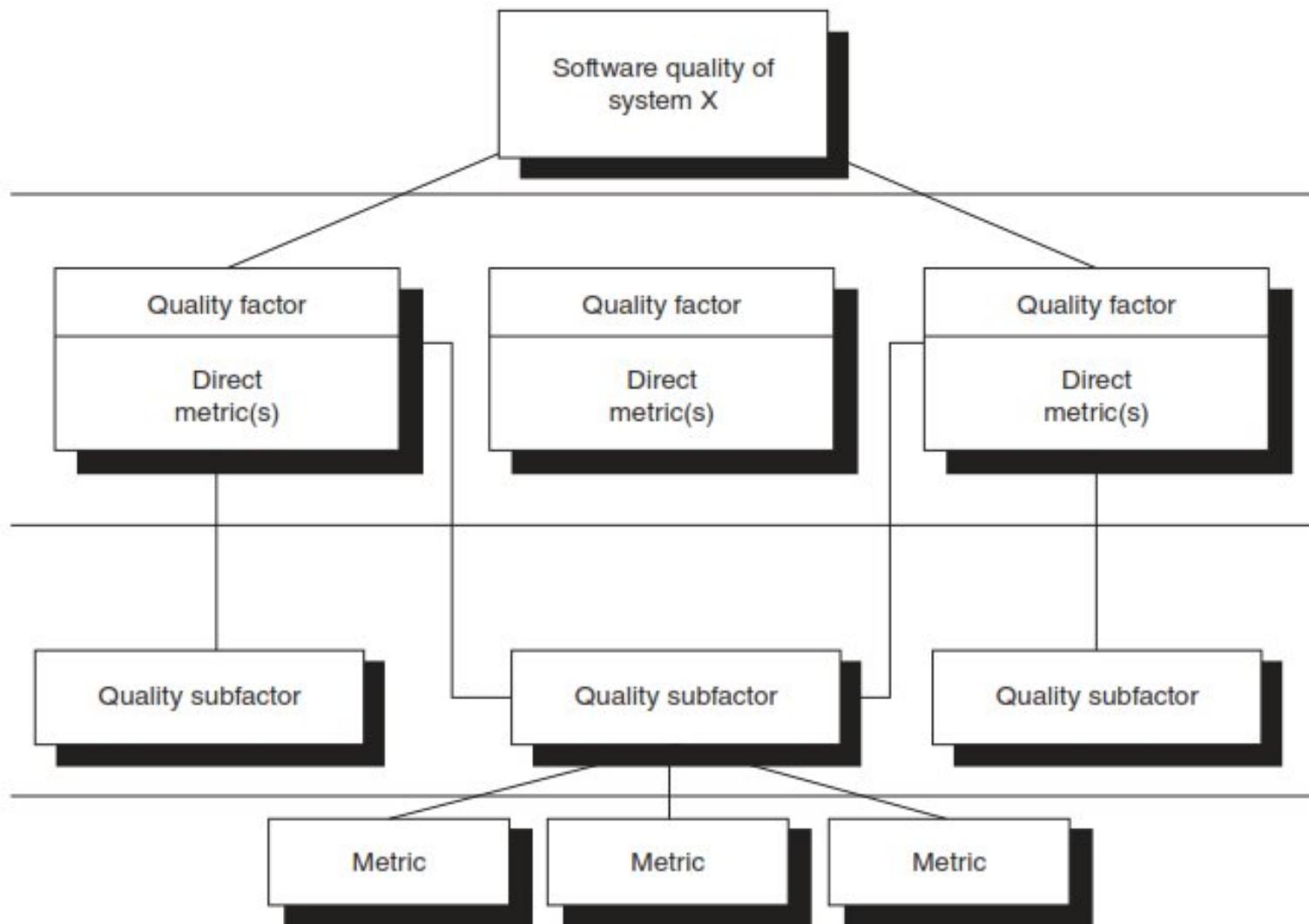


Figure 3.3 Framework for measuring software quality as per the IEEE 1061 [IEE 98b].

-
- 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.

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 agreed-upon specifications, corrective actions should be taken by the developer)

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
-

Set of Standards

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

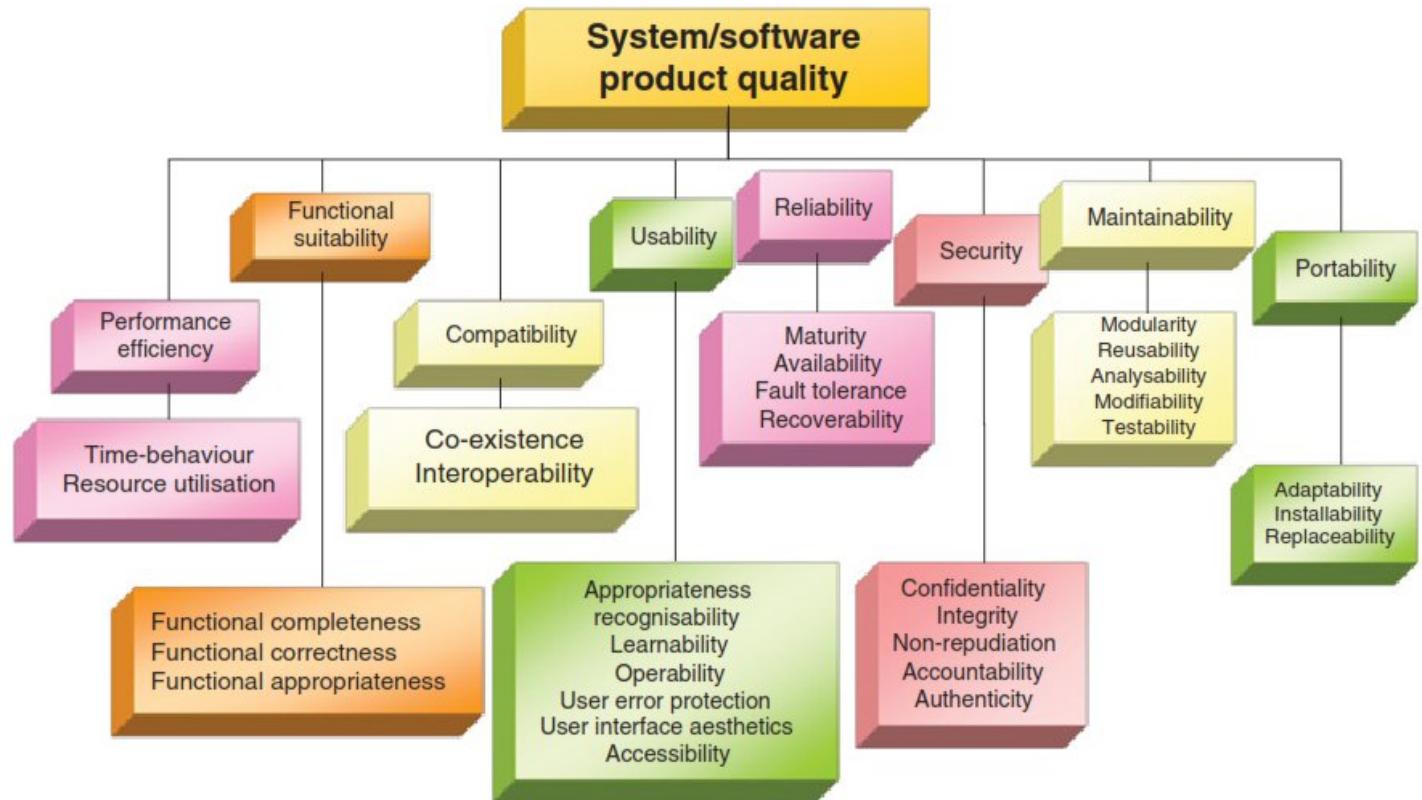
- (1) assess the quality of the development process,
- (2) assess the quality of the final product.

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

- Modularity
- Reusability
- Analyzability
- Modifiability
- Testability

Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers

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

Degree to which an asset can be used in more than one system, or in building other assets

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

Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality

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

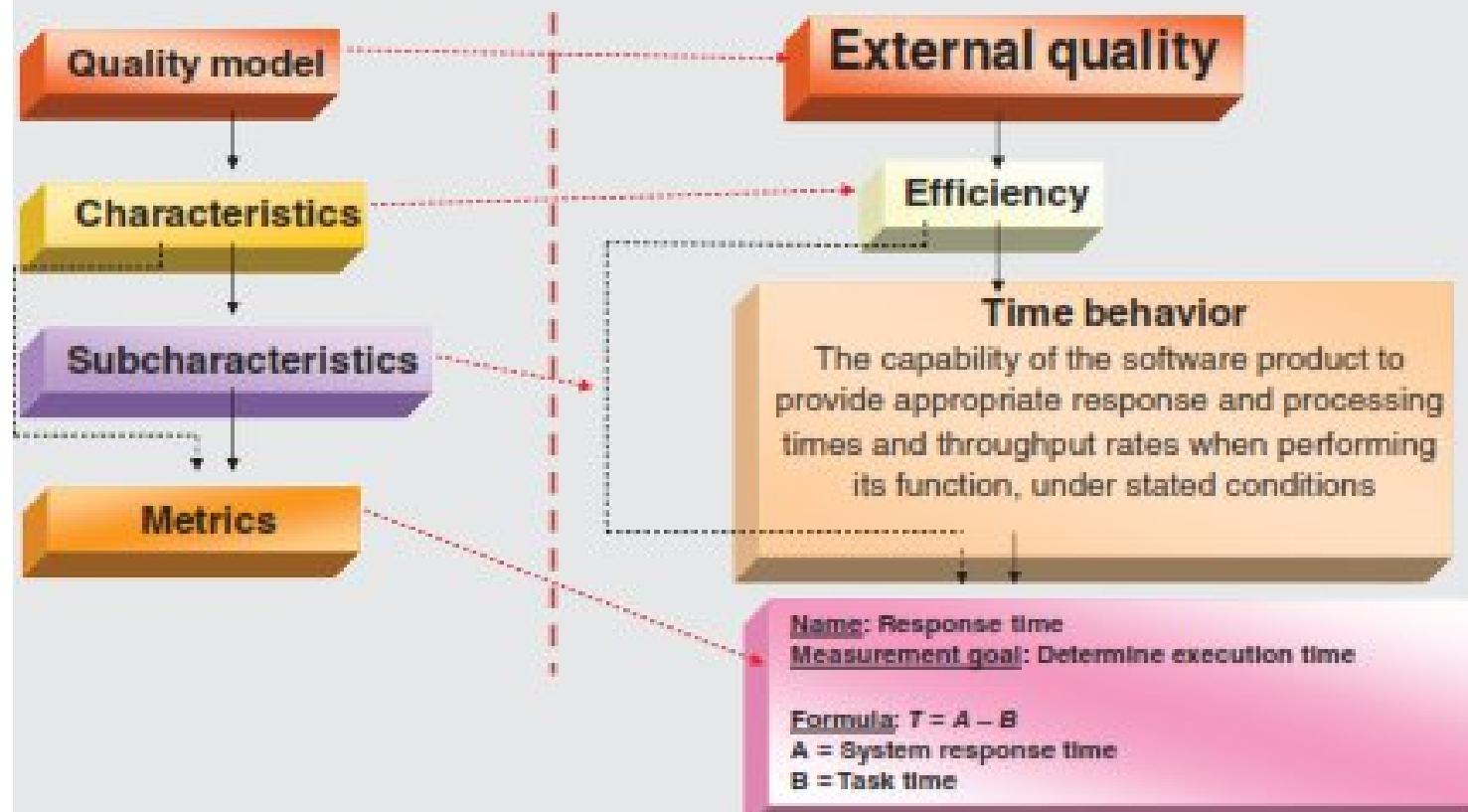


Evaluating Software Quality with the ISO 25010 Model

eve

lead

The quality model proposed by ISO 25010 is similar to the IEEE 1061 [IEE 98b] model. Choose a quality characteristic to evaluate—efficiency is used in the following example. Then choose one or more quality sub-characteristics to be evaluated (time behavior has been chosen in the following example). The last step is to clearly specify a measurement so that there is no possible misinterpretation of its result. A dotted line indicates that the sub-characteristics can be bypassed if necessary.



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.
-

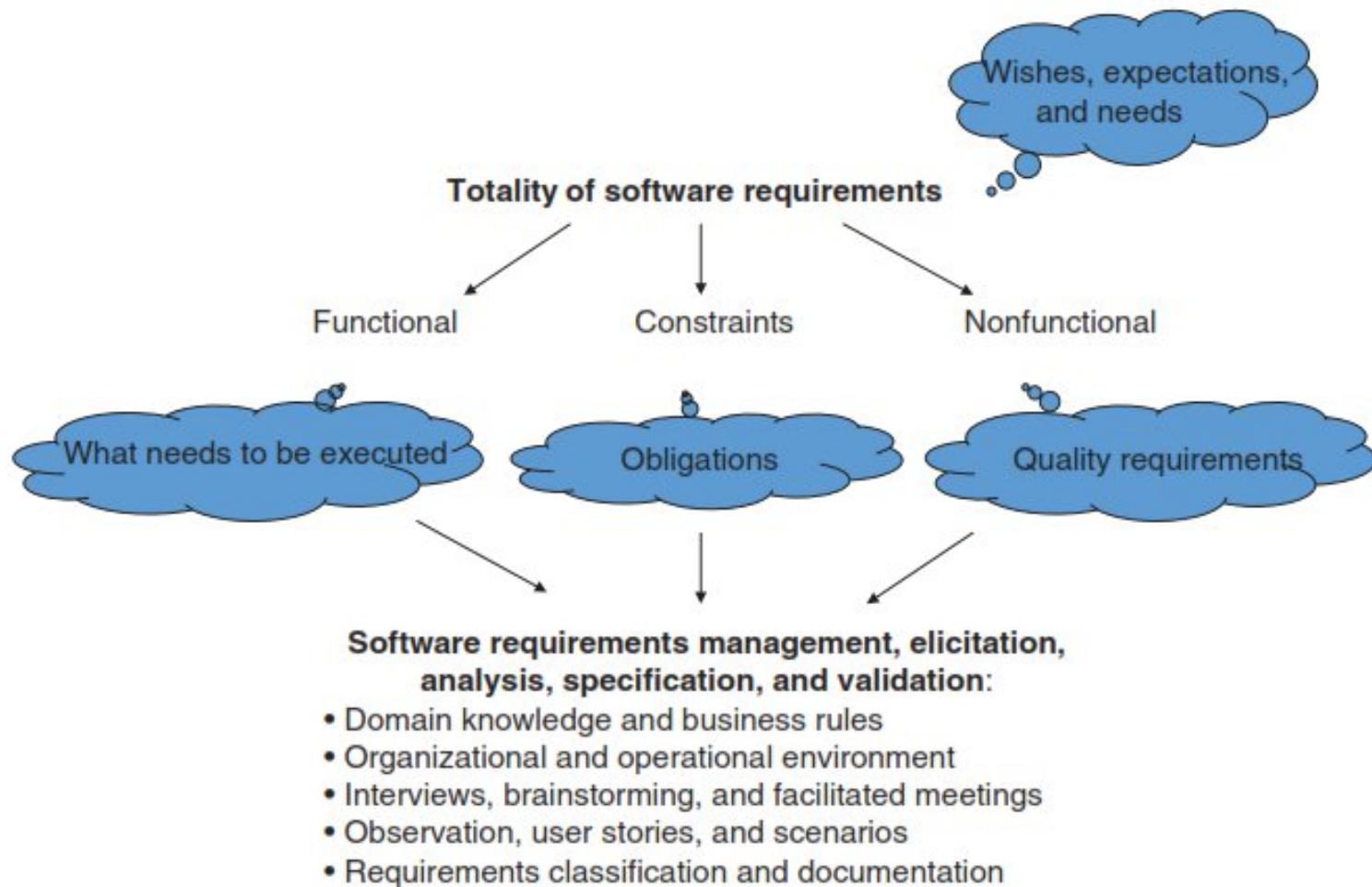


Figure 3.5 Context of software requirements elicitation.

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.



THANK YOU



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan





SE ZG501

Software Quality Assurance and

Testing

Lecture No. 3

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.

Specifying Quality Requirements: The Process

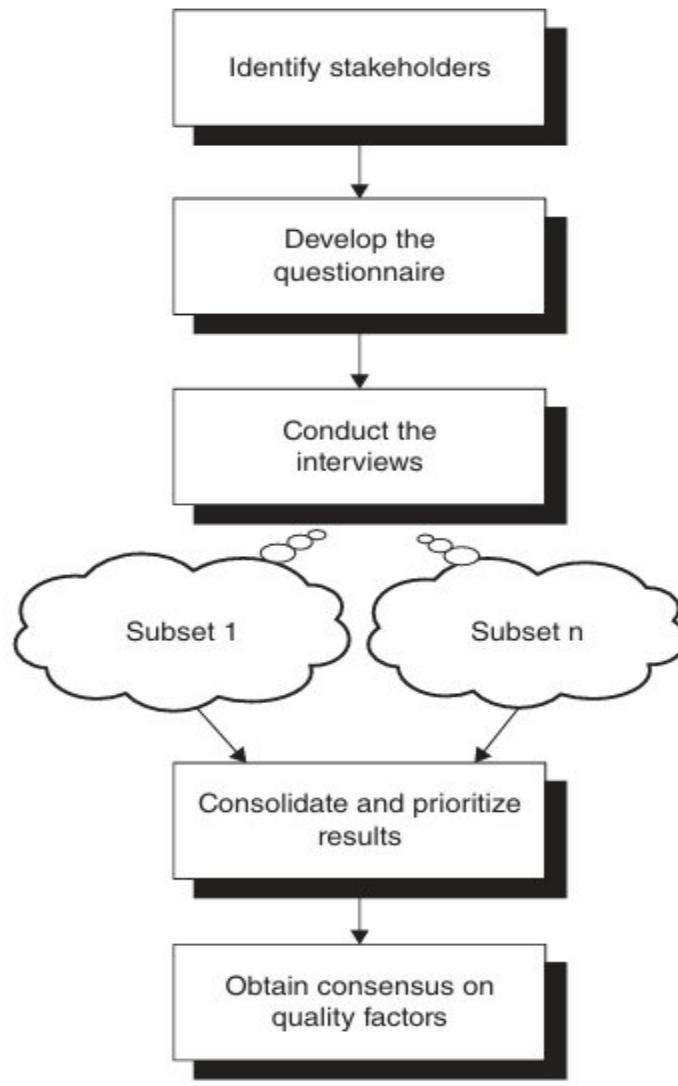


Figure 3.6 Steps suggested for defining non-functional requirements.

-
- 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 measures 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.

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.

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

$$\sigma = E \cdot \epsilon$$

Newton's Law

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

Boyle-Mariotte's Law

$$p_1 x V_1 = p_2 x V_2$$

Curie's Law

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

Refraction Law

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

Gravitational Law

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

Ohm's Law

$$V = RI$$

Coulomb's law

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

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 serves as the framework for developing and approving standards, including international ISO standards and those from professional organizations like 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).

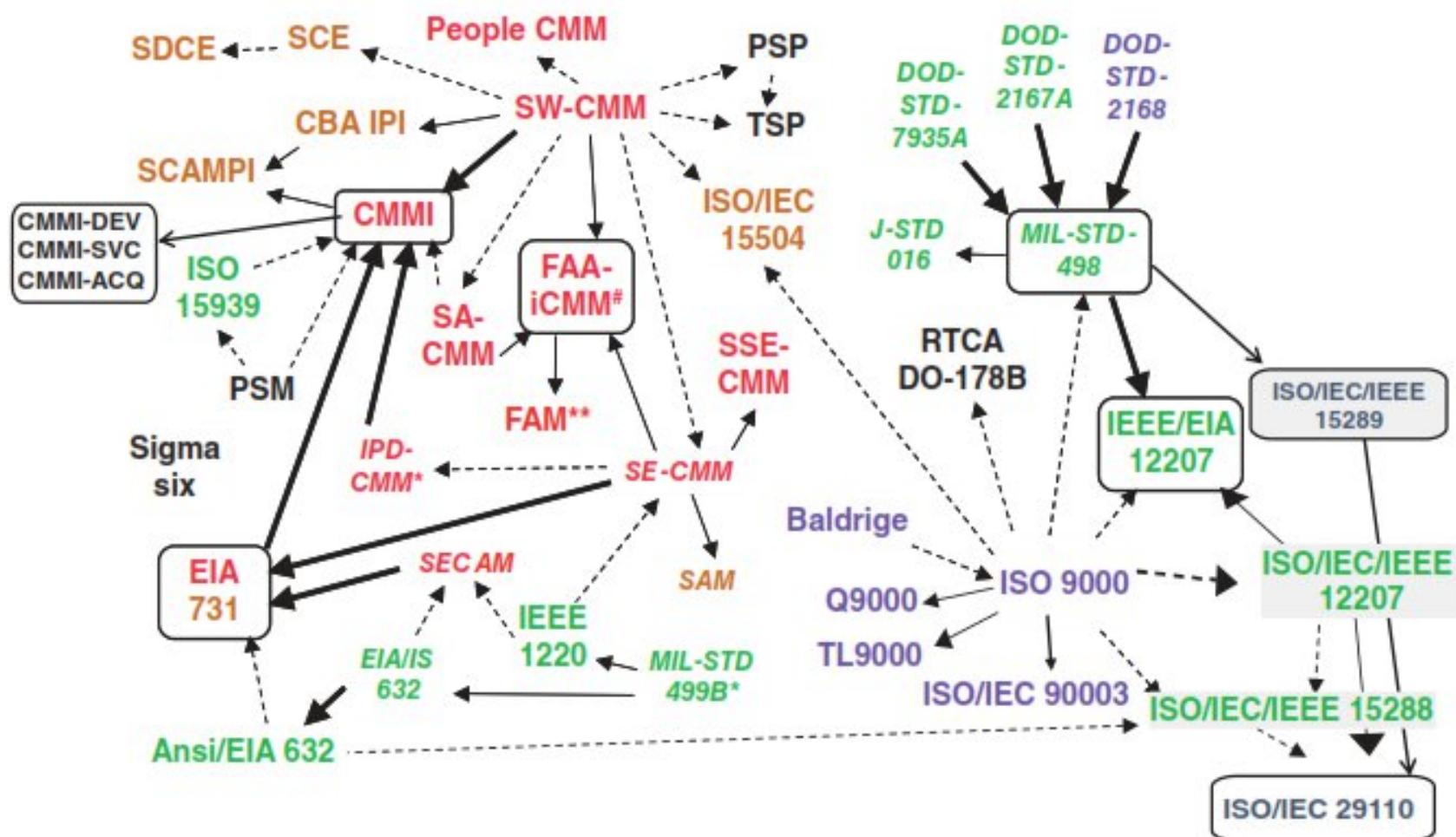


Figure 4.2 The development of standards and models.

- 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:

The Continuous Evolution of Standards



Figure 4.3 The evolution of standards SC7 [SUR 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.



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
-

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.
-

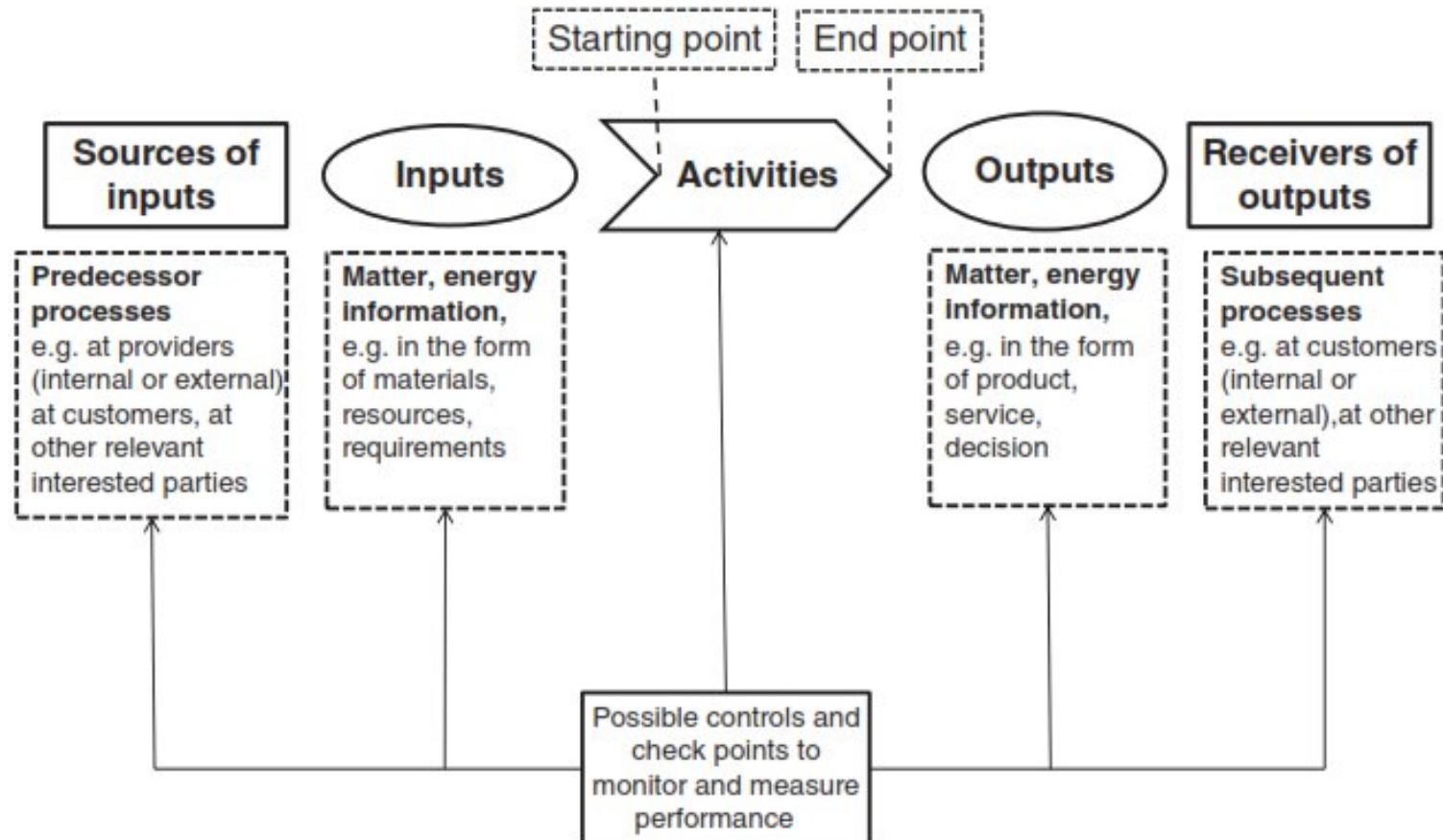


Figure 4.4 Elements of a process [ISO 15].

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.

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.

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, development, supply, 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.

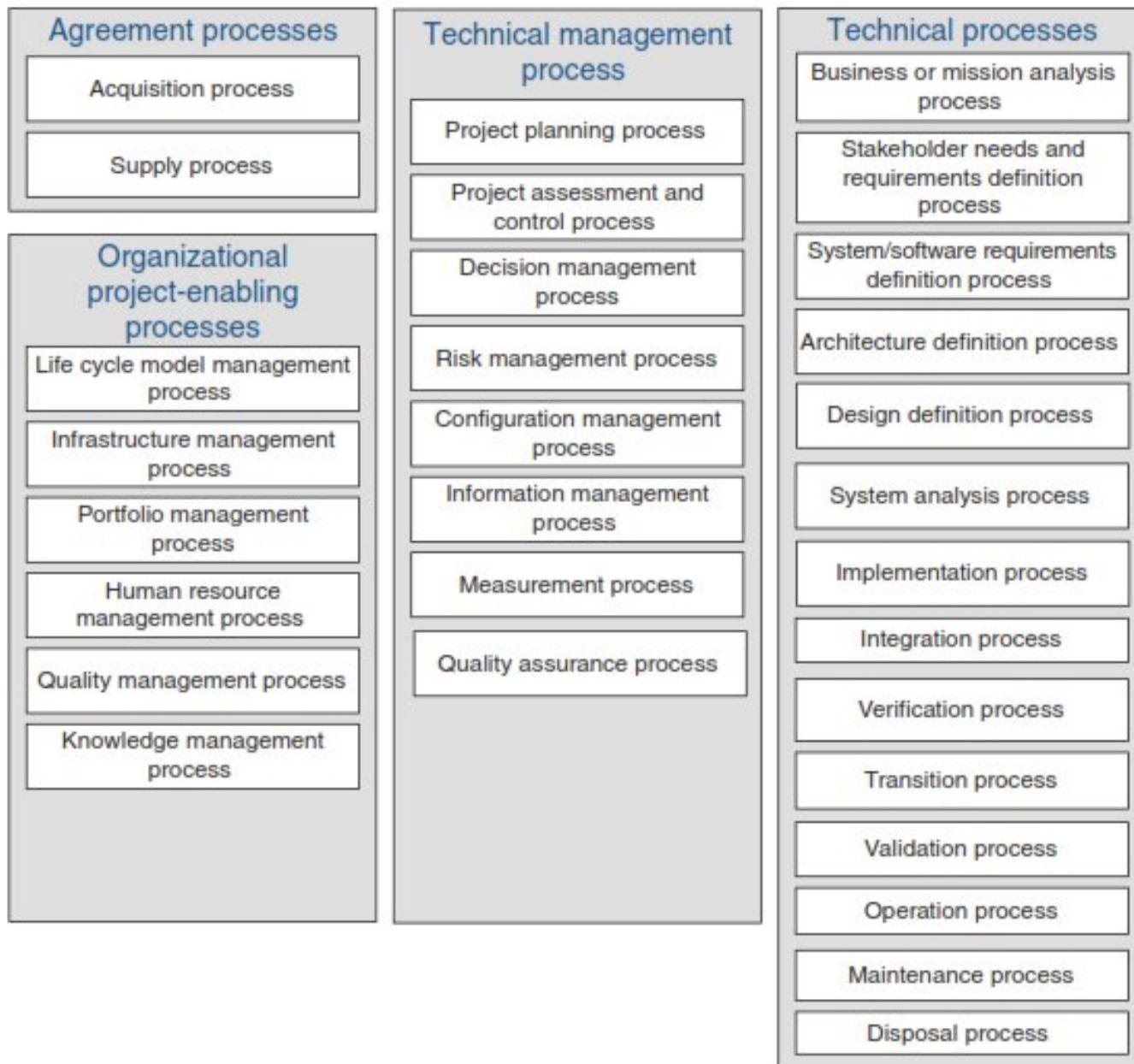


Figure 4.5 The four life cycle process groups of ISO 12207 [ISO 17].

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



For an organization: It aids in establishing a desired process environment, supported by appropriate methods, procedures, techniques, tools, and trained personnel.

For a project: It assists in selecting, structuring, and employing elements from established life cycle processes to deliver products and services effectively.

For an acquirer and a supplier: It facilitates the development of agreements concerning processes and activities, ensuring clarity and mutual understanding.

For organizations and assessors It serves as a process reference model for conducting process assessments, which can support organizational process improvement initiatives.

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.

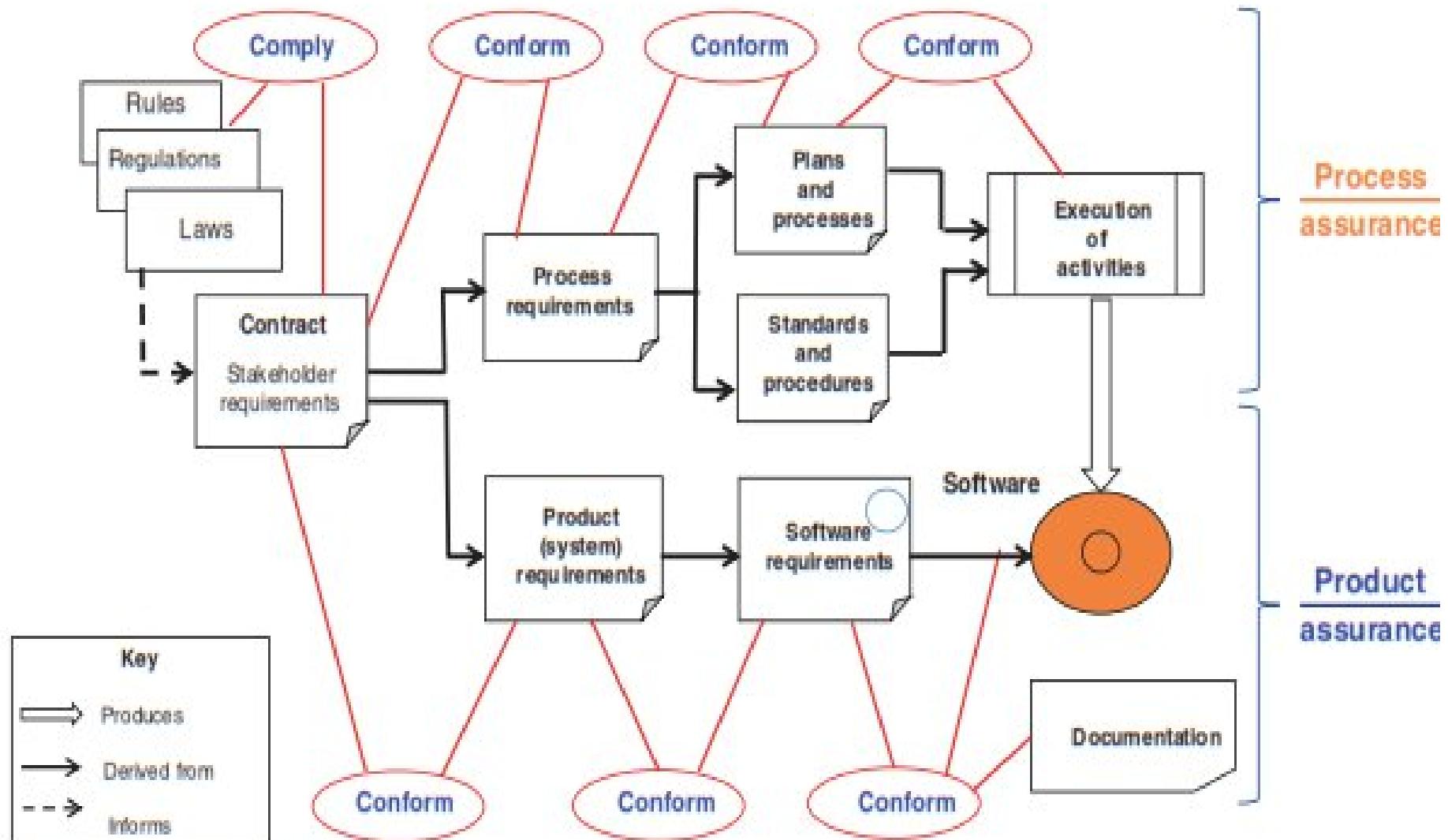


Figure 4.6 The links between requirements and the artifacts of a project [IEE 14].

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.

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.

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.

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.**

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) and the **CMMI for Acquisition** (CMMIACQ)

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.

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.

- 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

CMMI model structure.

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

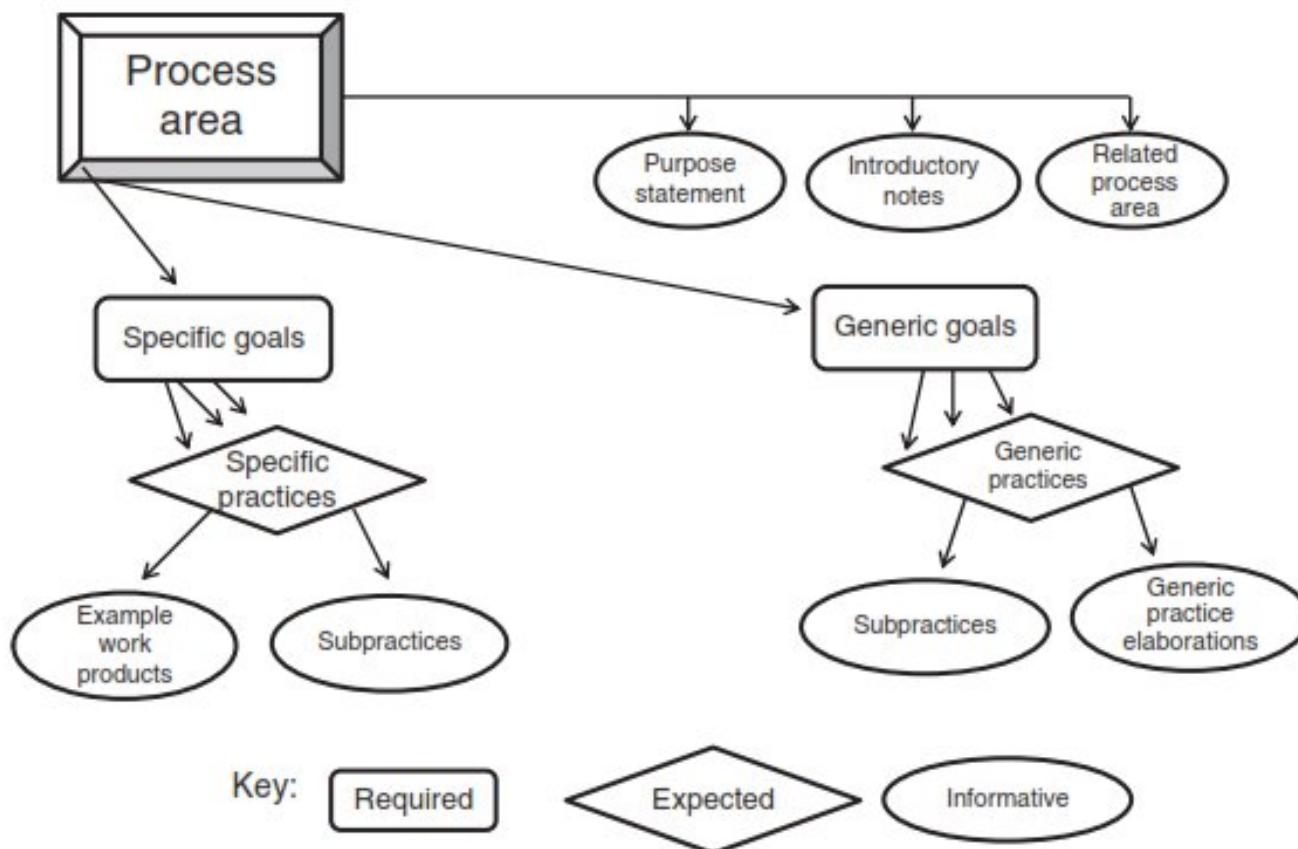


Figure 4.8 Structure of the staged representation of the CMMI [SEI 10a].

Level	Focus	Key process area	
5 Optimizing	<i>Continuous process improvement</i>	Organizational performance management causal analysis and resolution	Quality productivity
4 Quantitatively managed	<i>Quantitative management</i>	Organizational process performance quantitative project management	
3 Defined	<i>Process standardization</i>	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	<i>Basic project management</i>	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.

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.

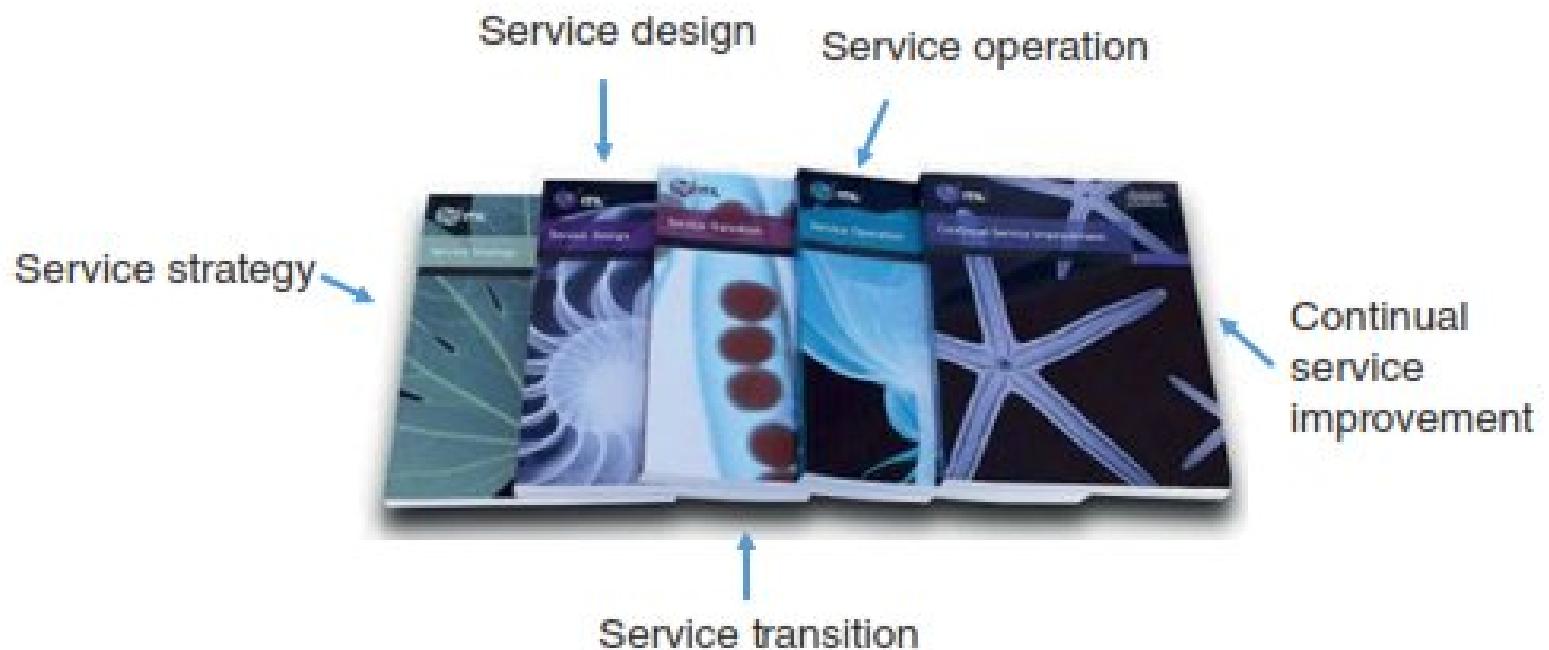


Figure 4.11 The main ITIL guides.

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.

Support center function

- Incident management
- Problem management
- Configuration management
- Change management
- Commissioning management

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

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)



THANK YOU



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS





SE ZG501

Software Quality Assurance and

Testing

Lecture No. 4

The need for a comprehensive definition of requirements



To 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*.

software

quality factors

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

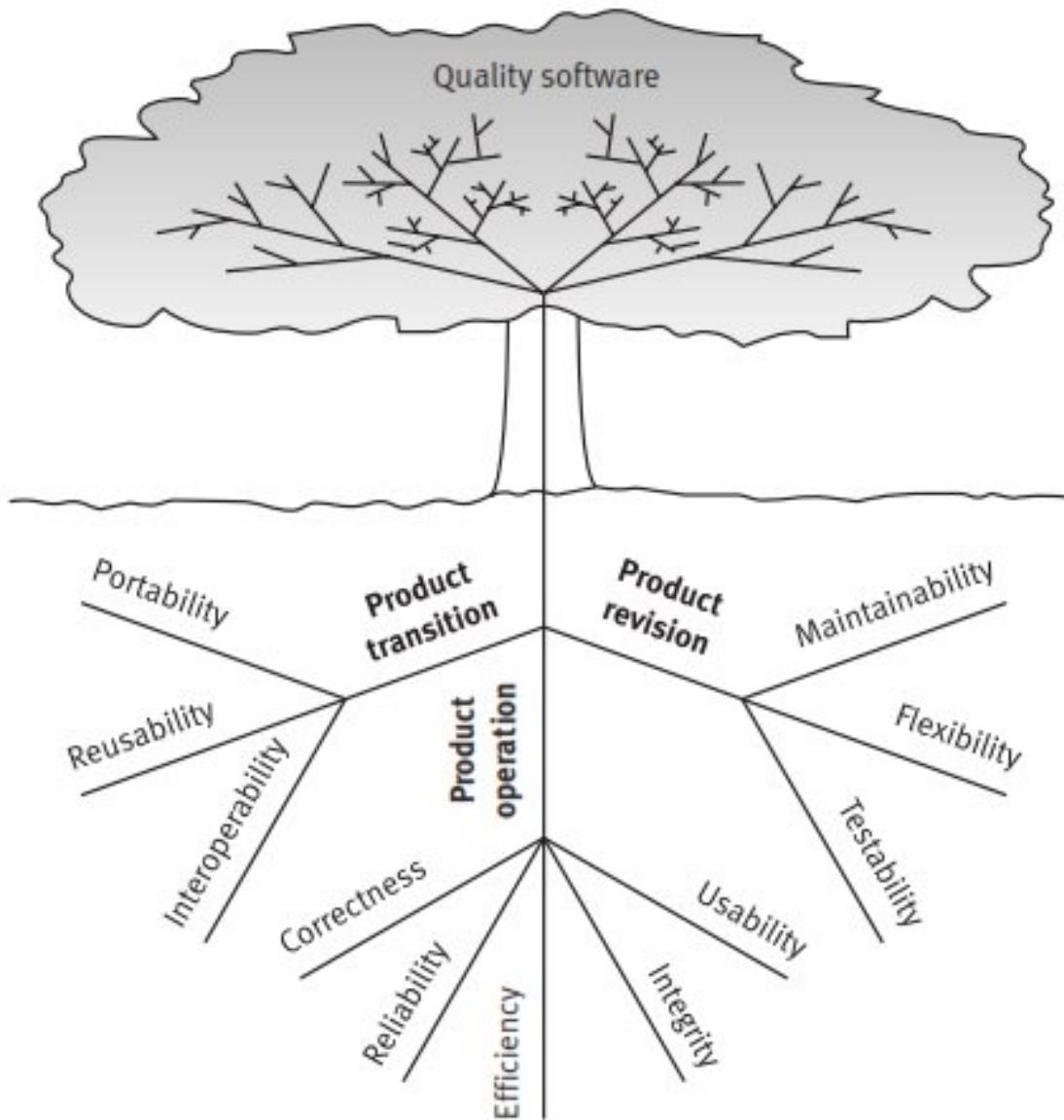
Subsequent models, consisting of 12 to 15 factors

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

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



Product operation software quality factors

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

For example:

- Displaying a customer's balance in the sales accounting information system.
- Regulating air supply as a function of temperature, as specified by the firmware of an industrial control unit

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.

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

Flexibility in software means how easily it can be updated or adjusted with minimal effort. It includes adapting the software for different customers, scales of operation, or product ranges in the same industry with minimal resources like time or effort.

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 refers to how easily an information system can be tested during development and operation. Testability requirements focus on **features that make testing easier**, such as:

- Predefined intermediate results to verify specific parts of the system.
- Log files to track system behavior and identify issues.

These features help testers quickly find and fix problems.

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.

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.

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).

Comparison of the alternative models

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

- The Evans and Marciak 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			+

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.

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.

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).

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**.

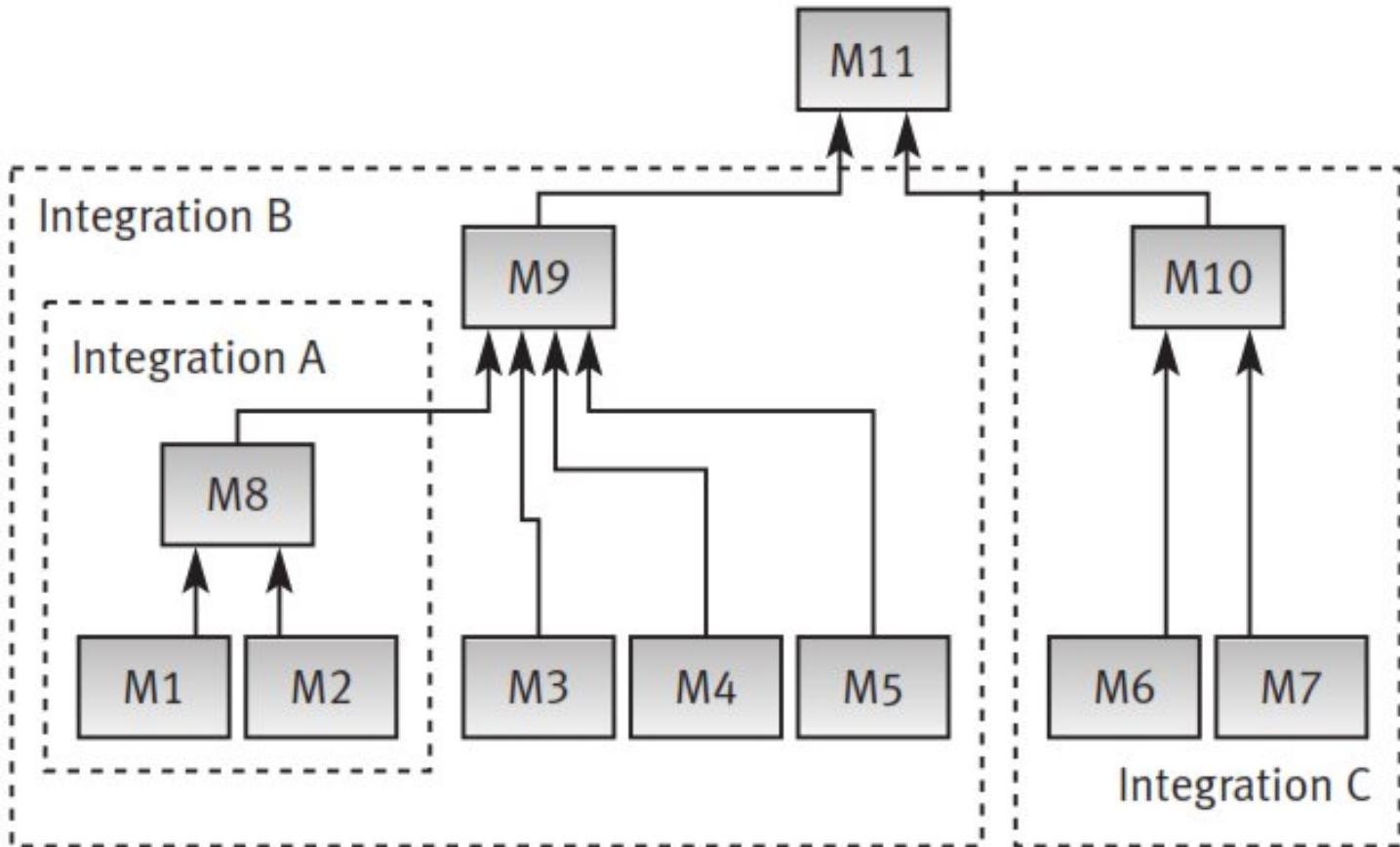
Both incremental testing strategies assume that the software package is constructed of a hierarchy of software modules.

Stage 4

Stage 3

Stage 2

Stage 1



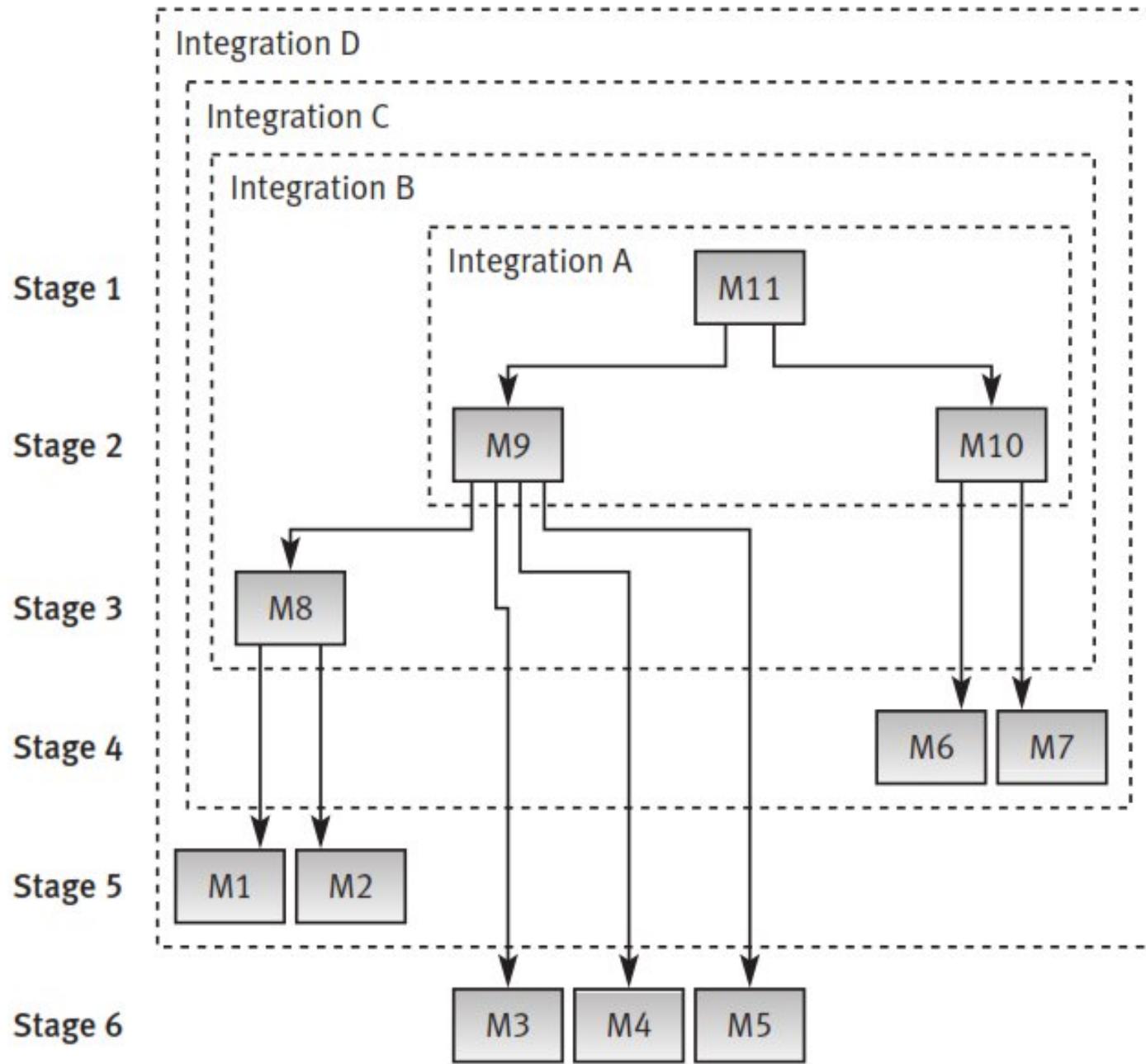
(a) Bottom-up testing

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.

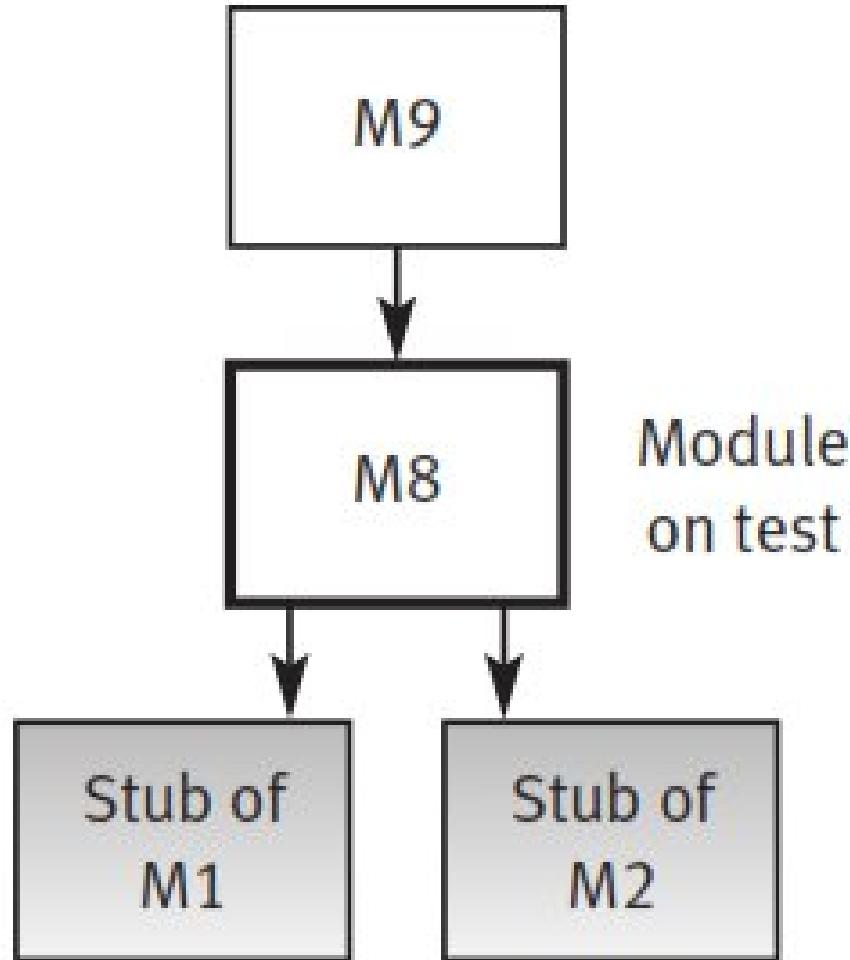
Stubs and drivers for incremental testing

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.

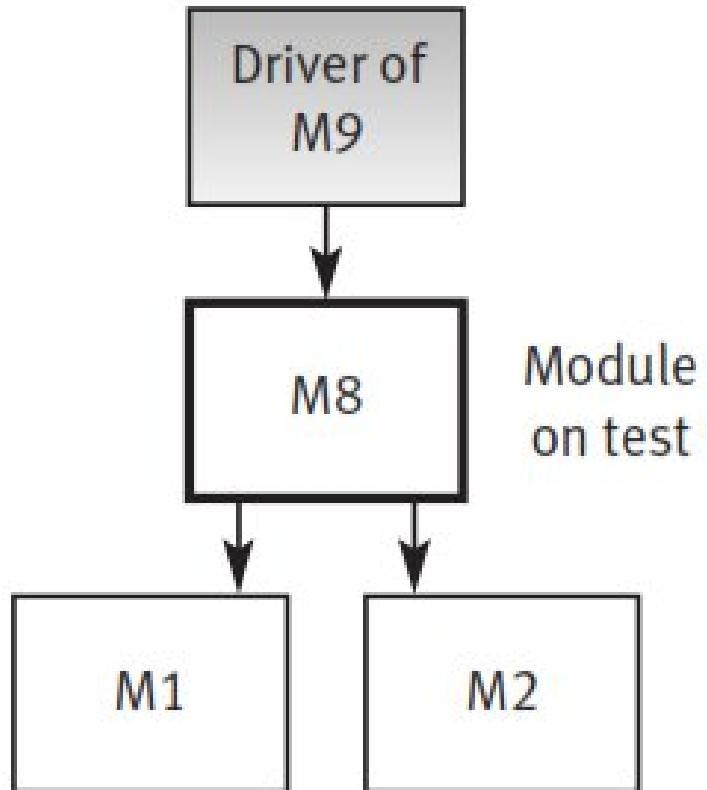
(a) Implementing top-down tests (Stage 3 testing of the example shown in Figure 9.1)



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)



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: 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)..

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.”
-

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?**

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.

Software test classifications

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.

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 1.2 Accuracy and completeness of documentation 1.3 Availability (reaction time) 1.4 Data processing and calculations correctness 1.5 Coding and documentation standards	1.1 Output correctness tests 1.2 Documentation tests 1.3 Availability (reaction time) tests 1.4 Data processing and calculations correctness tests 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	9. Portability tests 10. Reusability tests 11.1 Interoperability with other software 11.2 Interoperability with other equipment 11.1 Software interoperability tests 11.2 Equipment interoperability tests



THANK YOU



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS





SE ZG501

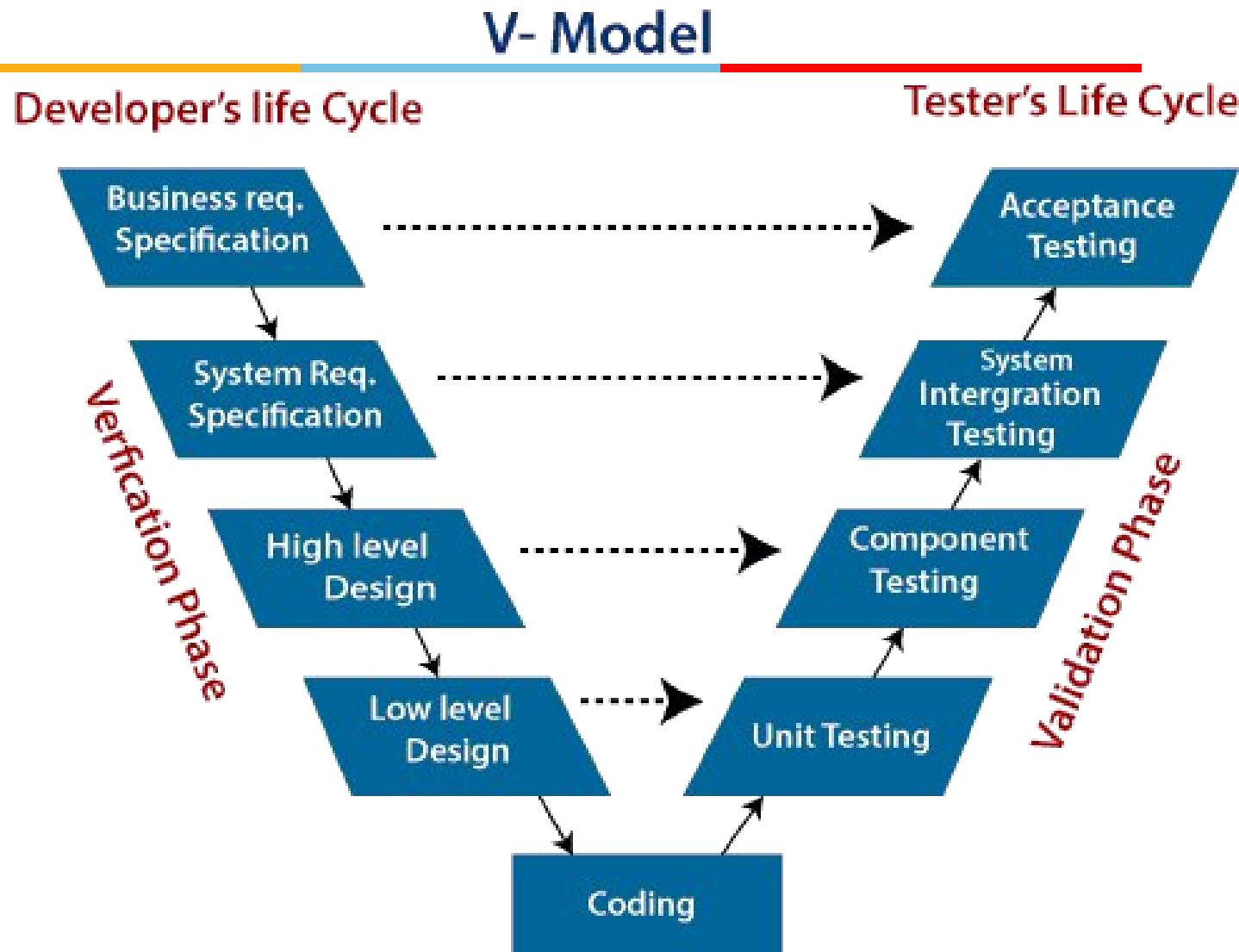
Software Quality Assurance and

Testing

Lecture No. 5

Deep driving SQA: Software Testing Techniques

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.**

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:

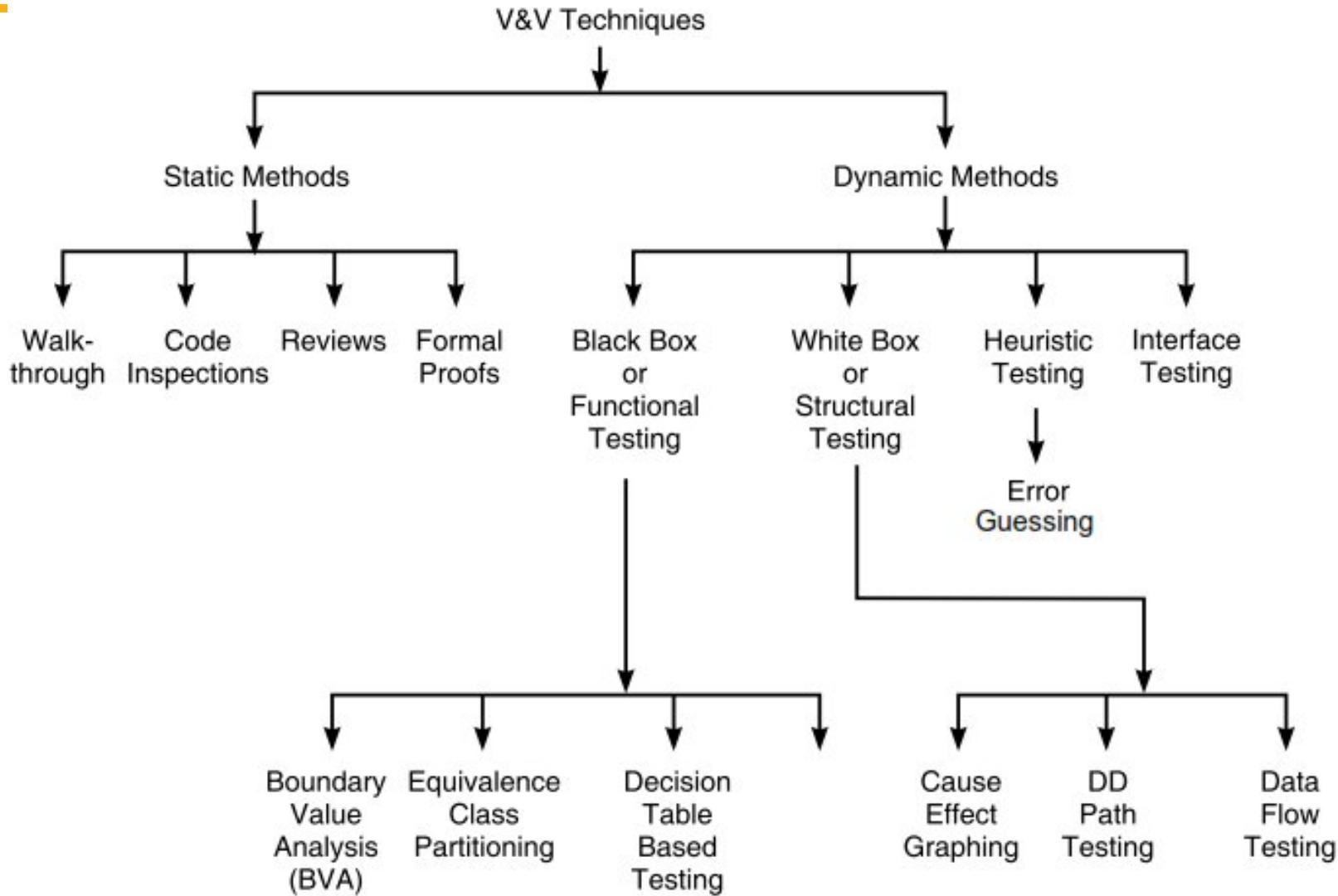
Preconditions: System is started.



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 transaction 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

CATEGORIZING V&V TECHNIQUES



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 but one variable to the extreme value and allowing the remaining variable to take the extreme value.

BVA is based upon a critical assumption that is known as **single fault assumption** theory.

The **single fault assumption** states that if an error occurs, it is due to a fault in only **one variable at a time**, while all other variables are kept constant at their extreme values.

For example, if a system has **n input variables**, when testing one variable, all other variables are held at their minimum or maximum values while the selected variable is tested at its boundary value

- For n variable to be checked:
 - **Maximum of $4n+1$ test cases**
-

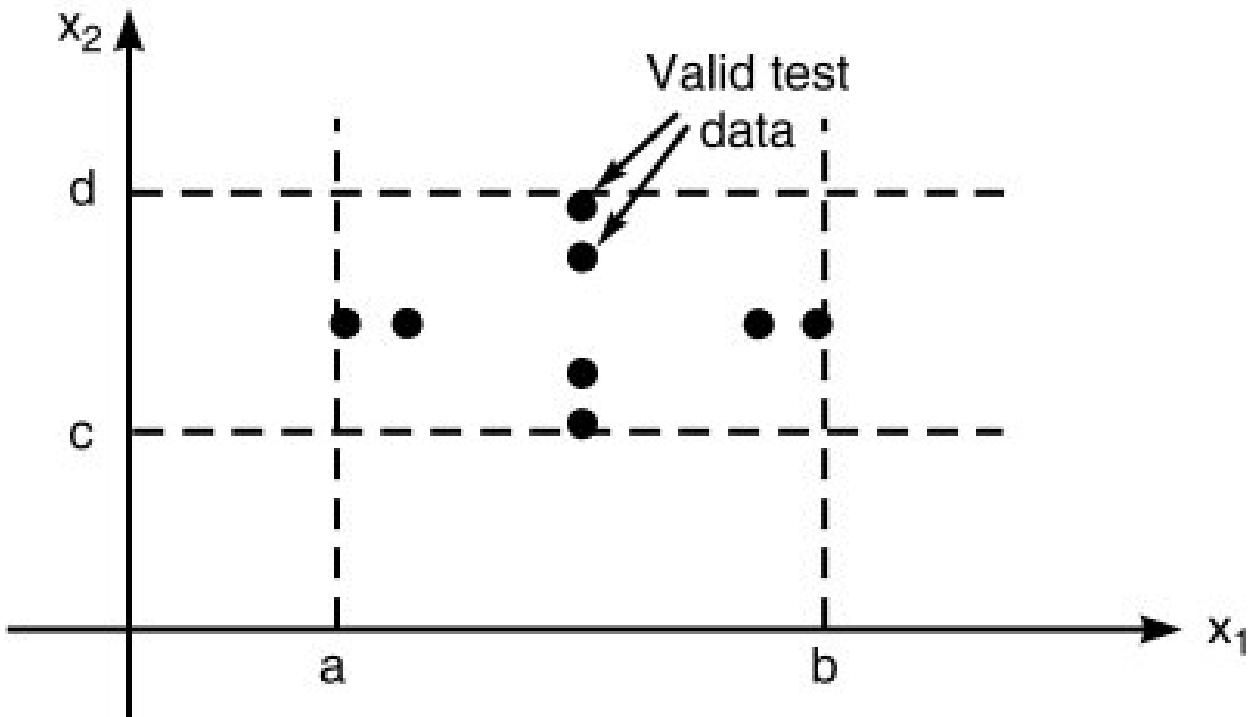


FIGURE 3.1 BVA Test Cases.

Problem: Consider a Program for determining the Previous Date.

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

$$1 \leq \text{Month} \leq 12$$

$$1 \leq \text{Day} \leq 31$$

$$1900 \leq \text{Year} \leq 2000$$

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.

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.

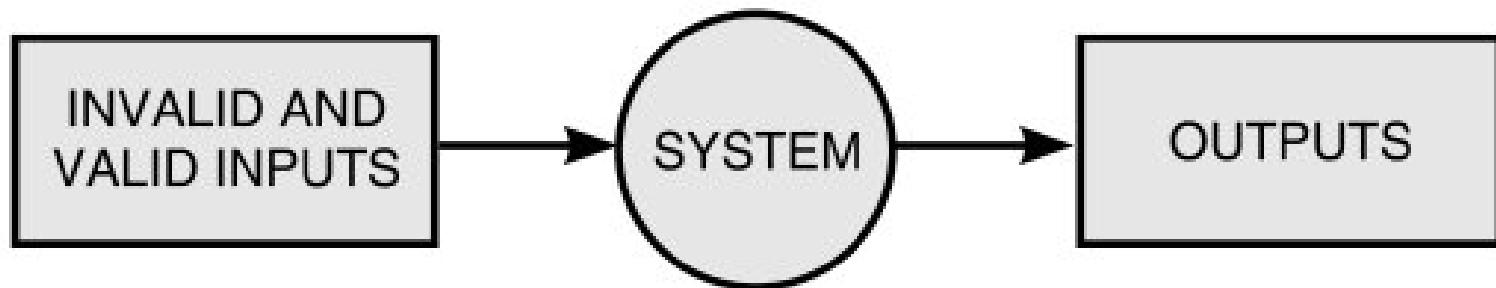


FIGURE 3.4 Equivalence Class Partitioning.

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 we 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 of each possible combinations of inputs.

Weak Robust Equivalence Class Testing: Like weak normal equivalence, weak robust testing **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.

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.

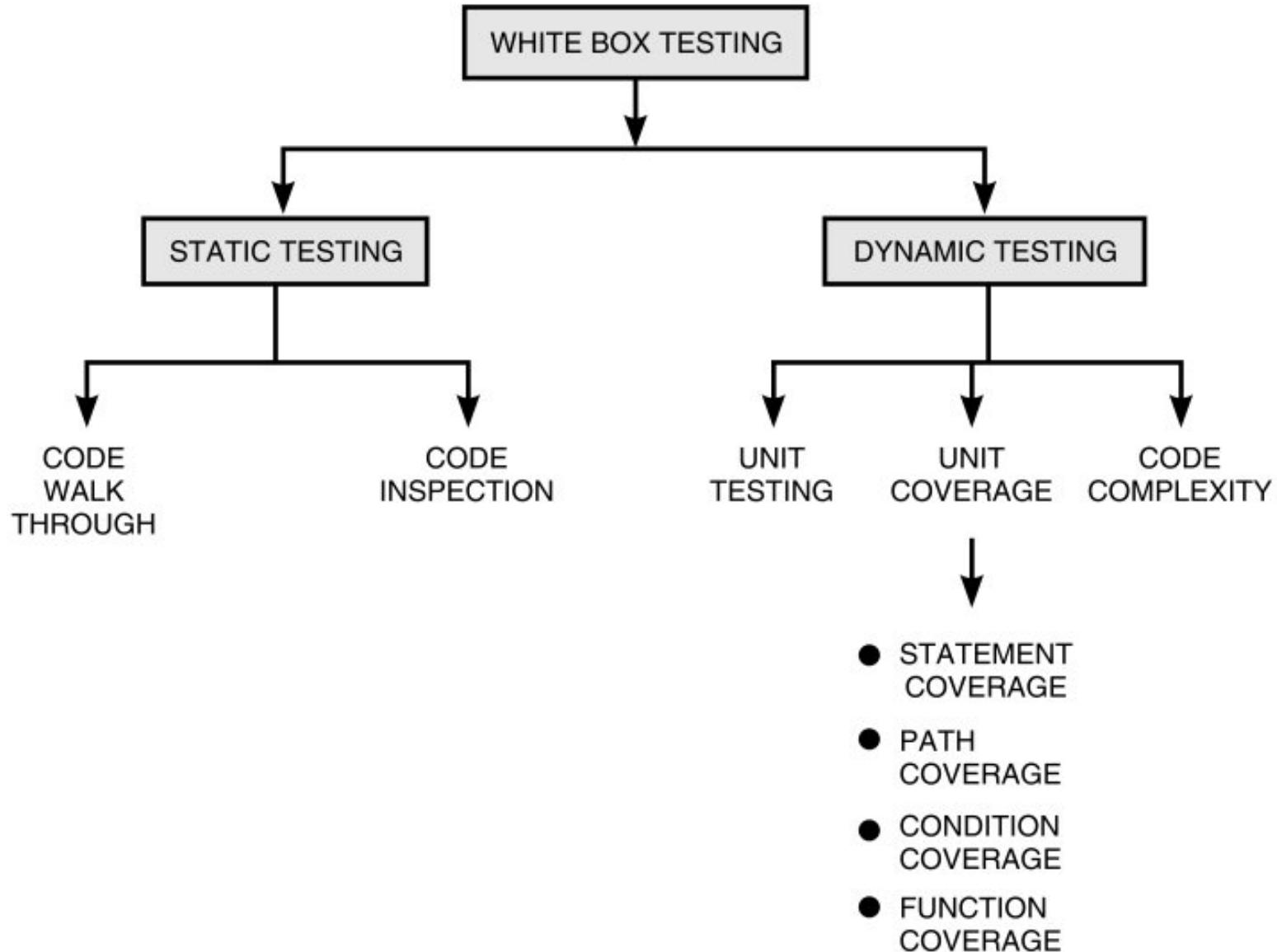


FIGURE 4.1 Classification of White-Box Testing.

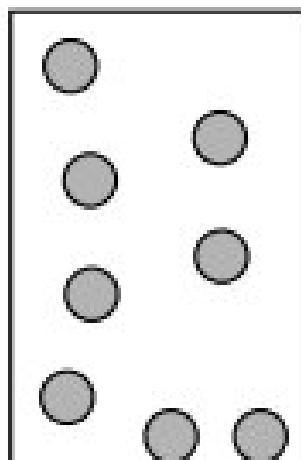
CODE COVERAGE TESTING

- 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***
-

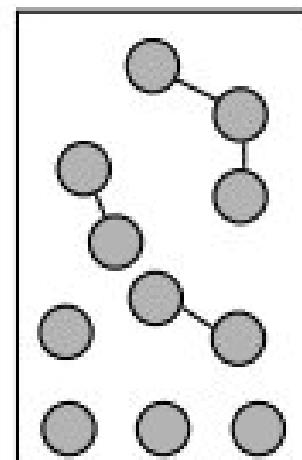
Test levels and types

Three levels of testing:

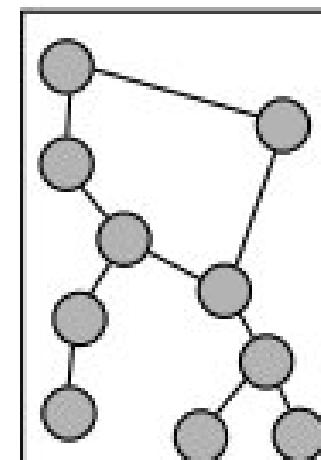
- 1. Unit testing**
- 2. Integration testing**
- 3. System testing**



UNIT TESTING

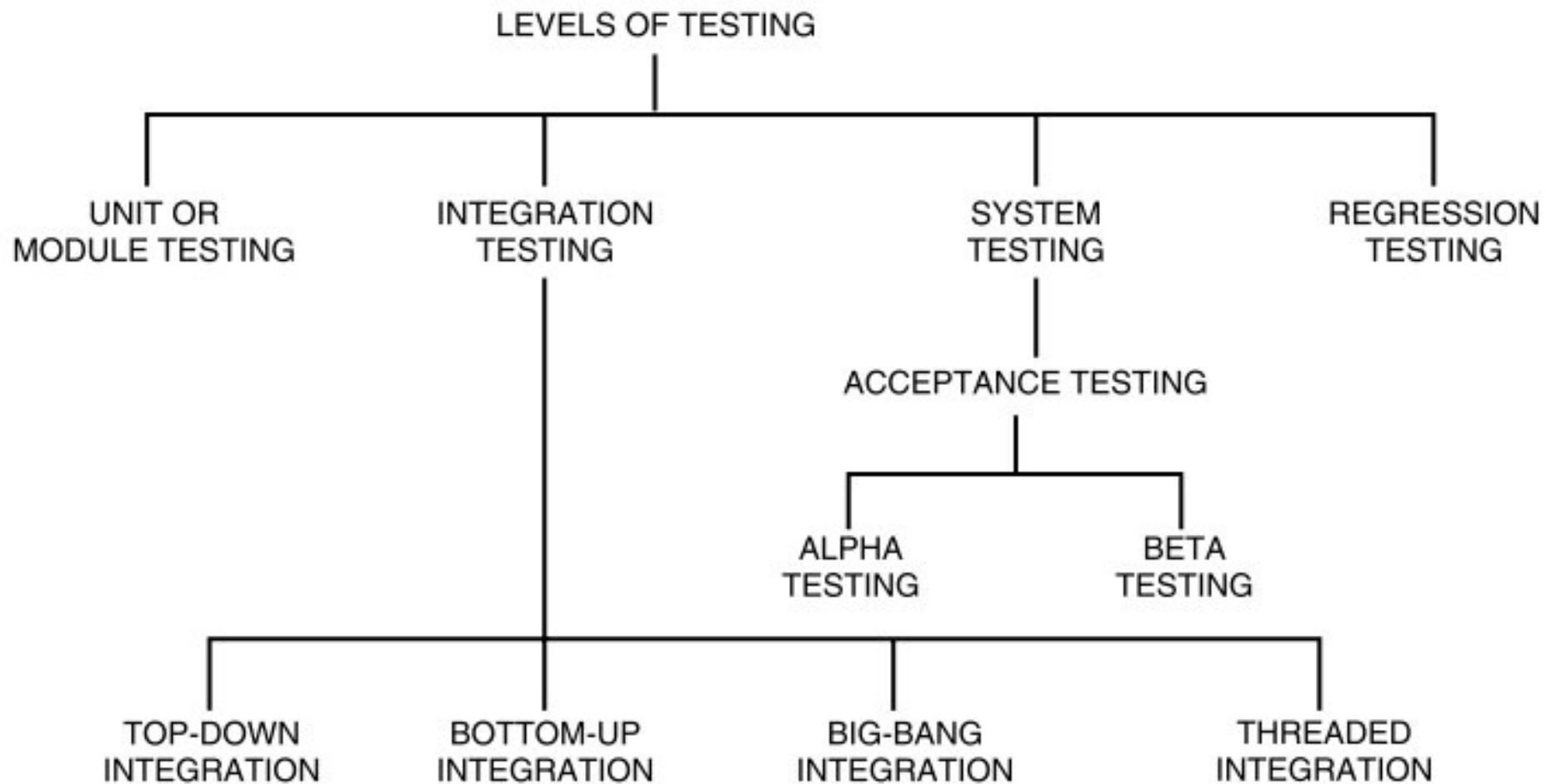


INTEGRATION TESTING



SYSTEM TESTING

FIGURE 7.1 Levels of Testing.



Unit (or module) testing

Unit (or module) testing “is the process of taking a module (an atomic unit) and running it in isolation from the rest of the software product by using prepared test cases and comparing the actual results with the results predicted by the specification and design module.”

It is a white-box testing technique.

Importance of unit testing:

1. Because modules are being tested individually, testing becomes easier.
2. It is more exhaustive.
3. Interface errors are eliminated.

INTEGRATION TESTING

- A system *is composed of multiple components or modules* that comprise hardware and software.
 - *Integration is defined as the set of interactions among components.*
 - *Testing the interaction between the modules and interaction with other systems externally is called integration testing.*
 - *The architecture and design can give the details of interactions within systems.*
-

Classification of integration testing

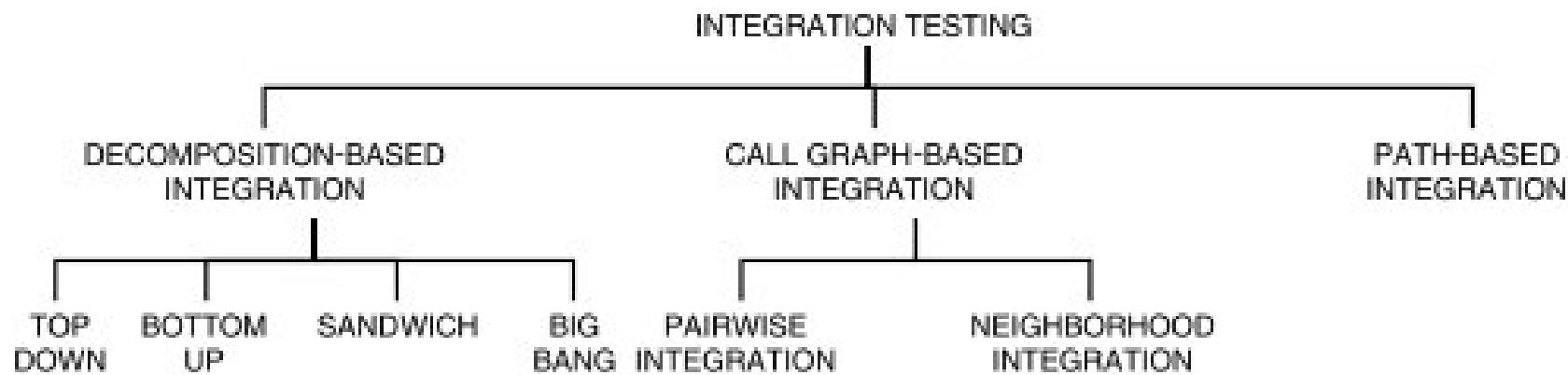


FIGURE 7.4

Decomposition-based Integration:

Decomposition-based integration involves functionally decomposing the system under test into a hierarchical structure, represented as a tree or in textual form. The primary objective is to evaluate the interfaces between individually tested units

Types Of Decomposition-based Techniques :Top-down Integration Approach



It begins with the main program, i.e., the **root of the tree**. Any lower-level unit that is called by the main program appears as a “stub.” A stub is a piece of throw-away code that emulates a called unit.

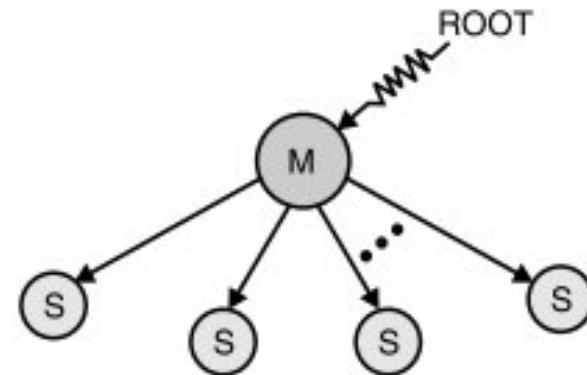


FIGURE 7.5 Stubs.

Bottom-up Integration Approach

It is a mirror image to the top-down order with the difference that **stubs are replaced by driver modules** that emulate units at the next level up in the tree.

start with the leaves of the decomposition tree and test them with specially coded drivers. Less throw-away code exists in drivers than there is in stubs.

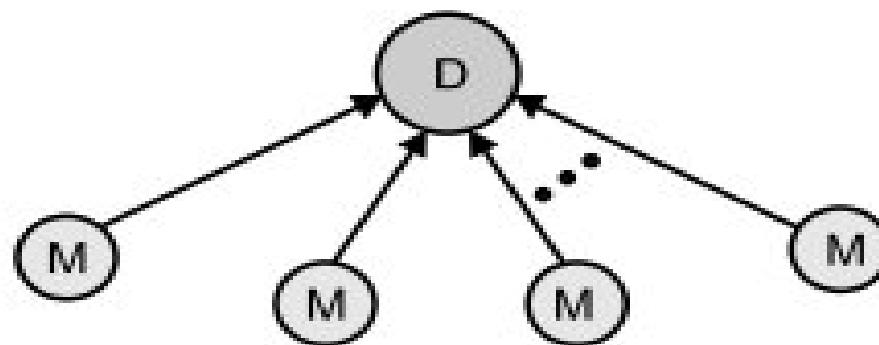


FIGURE 7.6 Drivers.

Sandwich Integration Approach

The **Sandwich Integration Approach**, also known as **Bidirectional Integration**, combines **Top-Down** and **Bottom-Up** integration testing methods.

- **Hybrid Approach:** It **merges top-down and bottom-up techniques** to accelerate integration.
- **Reduced Stub and Driver Effort:** Since both directions are integrated simultaneously, **fewer stubs and drivers are needed**.
- **Initial Testing with Stubs & Drivers:** Early integration relies on stubs (to replace lower modules) and drivers (to simulate higher modules).
- **Focus on Key Components:** Emphasis is placed on testing newly developed or critical components efficiently.

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*

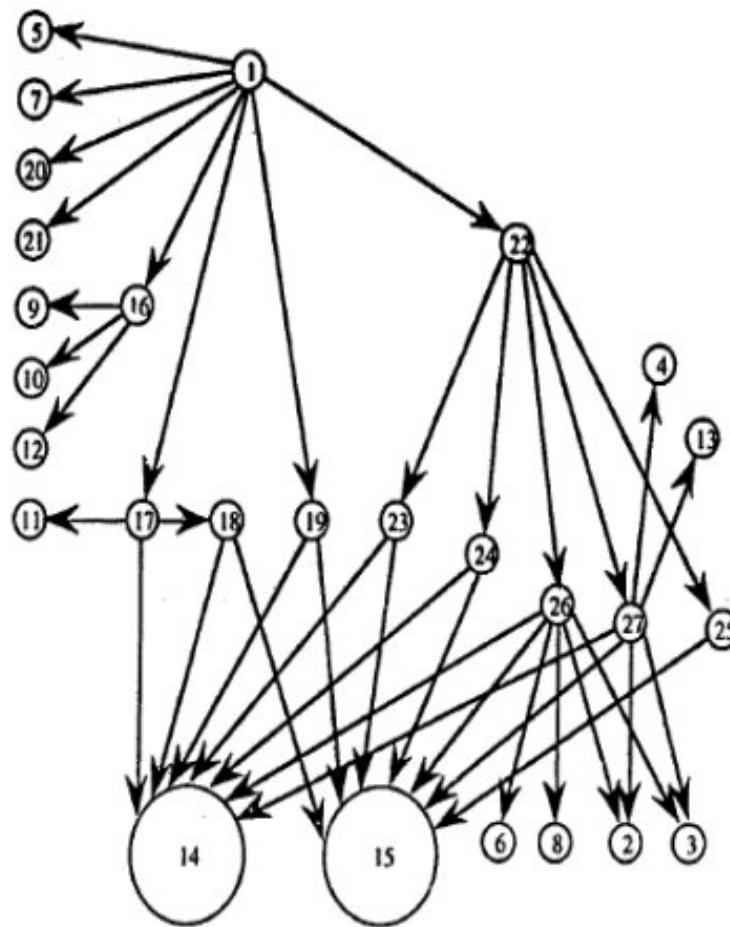
S. No.	Factors	Suggested method
1.	Clear requirements and design.	Top-down approach.
2.	Dynamically changing requirements, design, and architecture.	Bottom-up approach.
3.	Changing architecture and stable design.	Sandwich (or bi-directional) approach.
4.	Limited changes to existing architecture with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

Call Graph-based Integration

- Structural testing (shows **how functions/modules interact**)
- A **Call Graph** is a **directed graph** that represents **function/method calls** within a program.
- Call graph is a directed graph thus, we can use it as a program graph.
 - *Pairwise Integration*
 - *Neighborhood Integration*

Table 13.1 SATM Units and Abbreviated Names

<i>Unit Number</i>	<i>Level Number</i>	<i>Unit Name</i>
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



Pairwise Integration

- Pairwise Integration reduces the effort needed to create **stubs and drivers** by directly integrating **two related units at a time** from the **call graph**.
- Instead of developing temporary stubs and drivers, we test real code components that interact.
- Each integration focuses on a single pair of units, ensuring their functionality before moving to the next pair.
- The total number of test sessions remains similar to top-down or bottom-up approaches, but the effort in creating extra test code is significantly reduced.

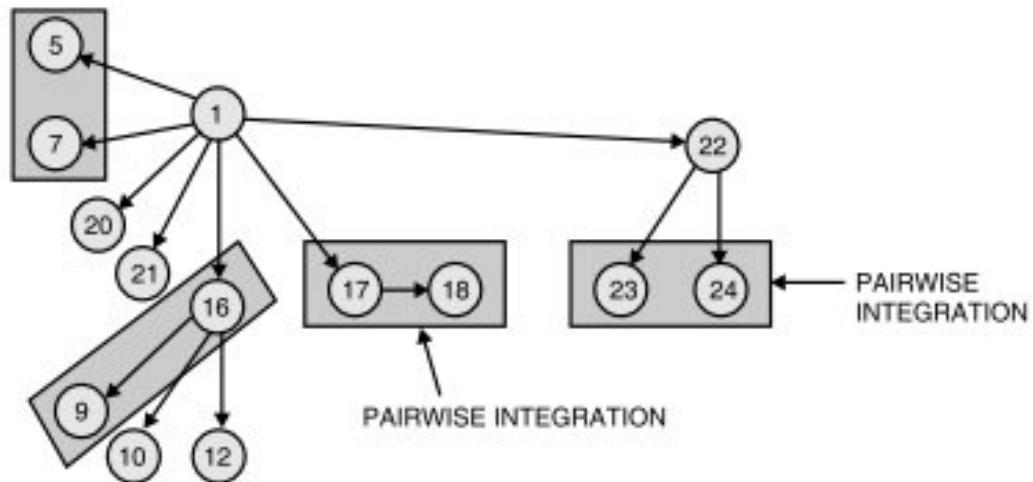


FIGURE 7.7 Pairwise Integration.

Neighborhood Integration

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

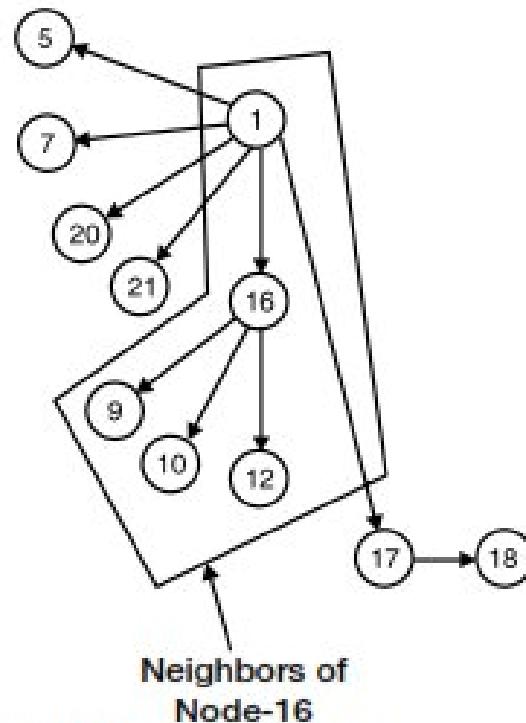


FIGURE 7.8 Neighborhood Integration.

Path-based Integration

- Path-Based Integration Testing is a **structural testing approach** that focuses on testing **all possible execution paths between interacting modules or components**. Instead of only checking individual interfaces, it ensures that different **execution flows and conditions are**
- The combination of structural and functional testing.
- structural and functional testing are 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, *cofunctioning* might be a good term.
- Interfaces are structural whereas interaction is behavioral.

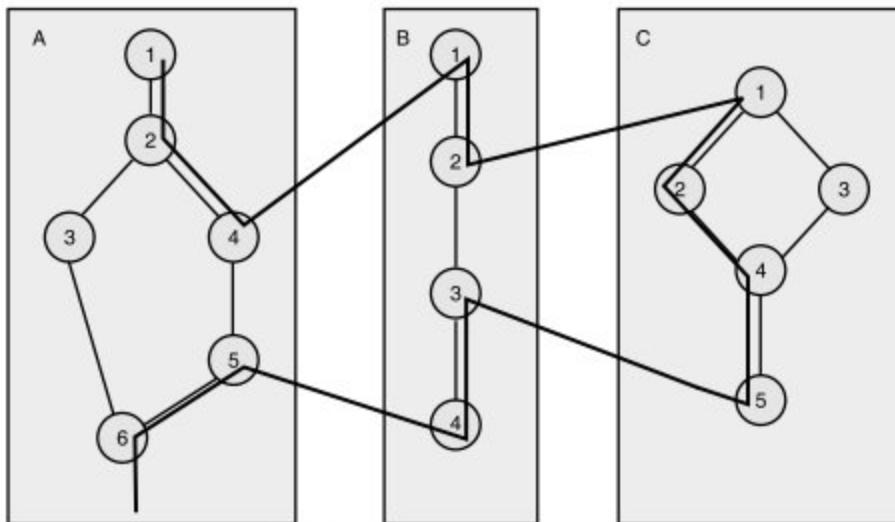


FIGURE 7.9 MM-Path Across Three Units (A, B, and C).

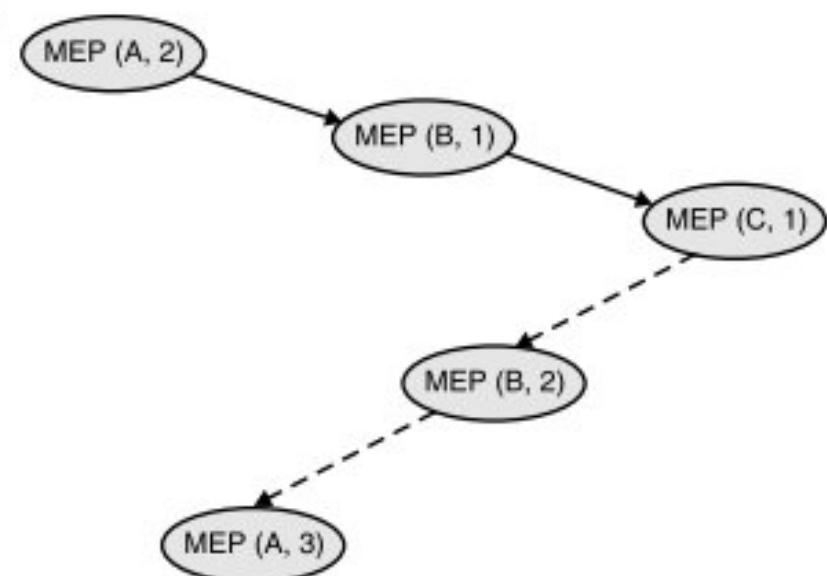


FIGURE 7.10 MM-Path Graph Derived from Figure 7.9.

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, and integration testing phases.

System testing is done to:

1. Provide independent perspective in testing as the team becomes more quality centric.
2. Bring in customer perspective in testing.
3. Provide a “fresh pair of eyes” to discover defects not found earlier by testing.
4. Test product behavior in a holistic, complete, and realistic environment.
5. Test both functional and non functional aspects of the product.
6. Build confidence in the product.
7. Analyze and reduce the risk of releasing the product.
8. Ensure all requirements are met and ready the product for acceptance testing.



Thank You



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS





SE ZG501

Software Quality Assurance and

Testing

Session 6

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:

1. Provide independent perspective in testing as the team becomes more quality centric.
2. Bring in customer perspective in testing.
3. Provide a “fresh pair of eyes” to discover defects not found earlier by testing.
4. Test product behavior in a holistic, complete, and realistic environment.
5. Test both functional and non functional aspects of the product.
6. Build confidence in the product.
7. Analyze and reduce the risk of releasing the product.
8. Ensure all requirements are met and ready the product for acceptance testing.

-
- An **independent test team** normally does system testing.
 - This independent test team is different from the team that does the component and integration testing.
 - The system test team generally **reports to a manager other than the product-manager to avoid conflicts** and to provide freedom to testing team during system testing.
 - Testing the product with an independent perspective and combining that with the **perspective of the customer** 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 <u>domain of the product</u> is required.	4. To do this type of testing, we need <u>domain, design, architecture</u> , and product's knowledge.
5. Configuration remains same for a test suite.	5. Test configuration is different for each test suite.

Test Execution Process

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.



Key components of a test plan

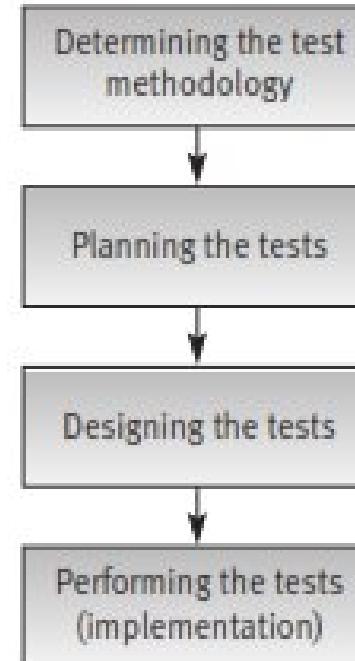
- Objectives,
- Scope,
- Test Items,
- Test Environment,
- Testing Strategy,
- Test Schedule,
- Resource Requirements,
- Risk Management
- Test Deliverables.

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 phase

The key challenges in testing methodology include :

- The appropriate required software quality standard
- The software testing strategy.

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

Key decisions to be made include:

- Selecting a testing approach: **Big Bang vs. Incremental**. If incremental, should it follow a **bottom-up or top-down** approach?
- Identifying which parts of the test plan should follow the **white-box testing** model.
- Determining which test cases should be executed using **automated testing**?

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?

Test planning documentation

The planning stage of the software system tests is commonly documented in a “software test plan” (STP).

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
-

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).

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**

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.

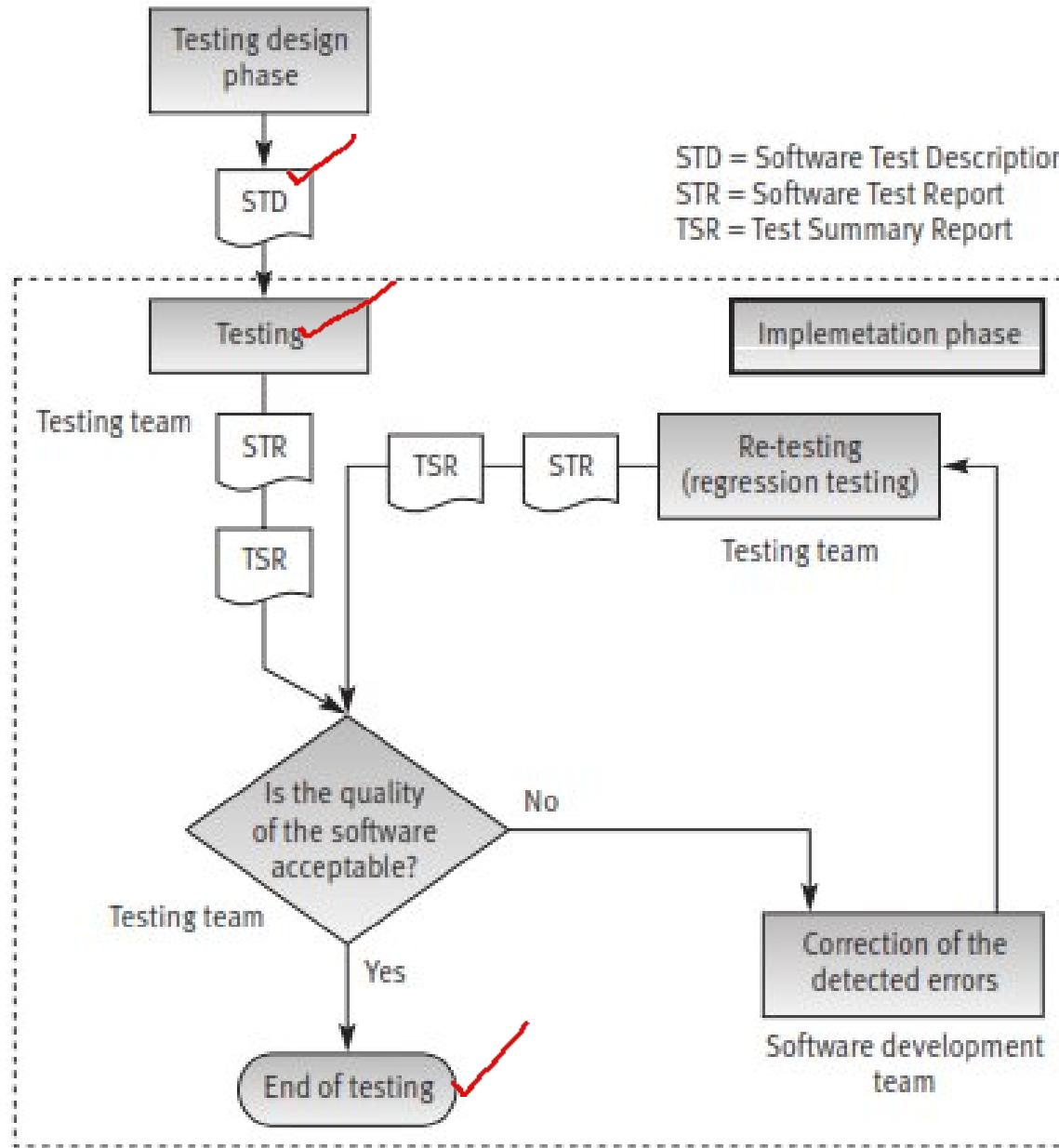


Figure 10.2: Implementation phase activities

The summary of the set of tests planned for a software package (or software development project) is documented in the “test summary report” (TSR).

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

Test case design

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

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.

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.

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.

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.**



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**

1. Accuracy and completeness of performance
2. Accuracy of results log and summary reports
3. Comprehensive information
4. Few manpower resources for test execution
5. Shorter testing periods
6. Performance of complete regression tests
7. Performance of test classes beyond the scope of manual testing

Disadvantages

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

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 over-the-counter sale (COTS).

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 in 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.

REGRESSION TESTING TECHNIQUE

- Regression testing is used to confirm that **previously fixed bugs remain resolved** and that **no new bugs have 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.

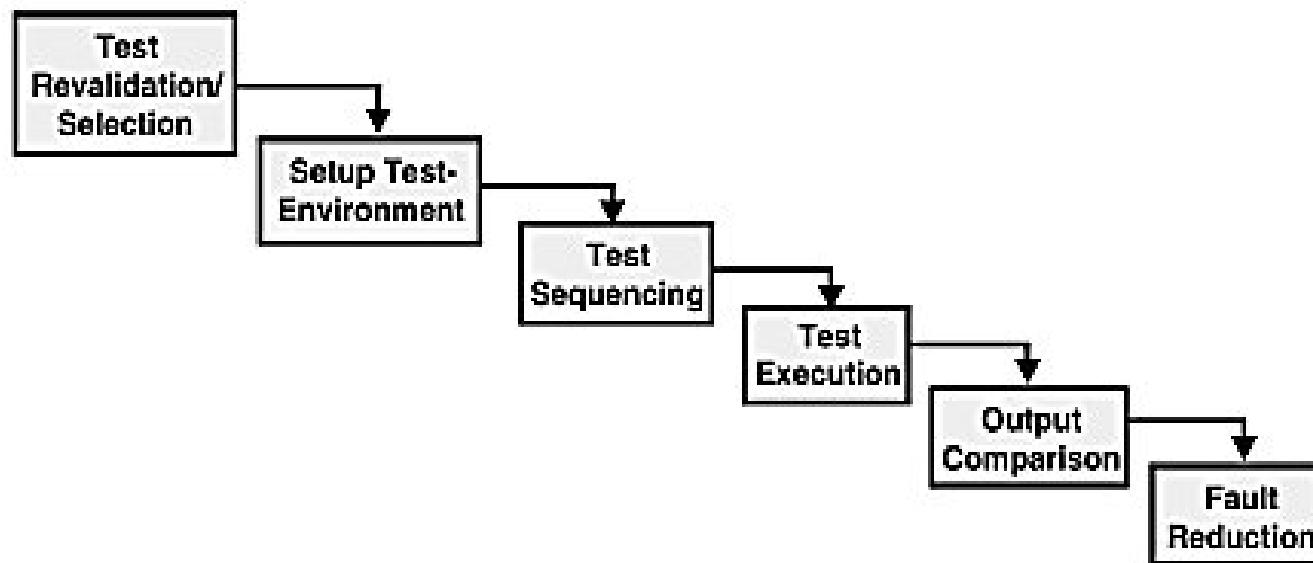


FIGURE 6.6

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

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.

Cost of quality

Quality costs = Prevention costs

- + Appraisal or evaluation costs
- + Internal and external failure costs
- + Warranty claims and loss of reputation costs

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
-

Informal reviews

An **informal review** lacks structure and standardization, leading to inconsistencies. Key issues include:

- **No documented process** – Different people conduct reviews in different ways.
- **Undefined roles** – No clear responsibilities for participants.
- **No specific objective** – Reviews don't focus on detecting faults.
- **Unplanned** – Conducted without proper scheduling.
- **No defect tracking** – Defects are not measured or recorded.
- **No management oversight** – Effectiveness is not evaluated.
- **No standards or checklists** – No guidelines to follow for identifying 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

Types of Reviews in Software Engineering:

1. Walk-through & Inspection:

1. Formal review methods to identify defects and improve quality.

2. Personal Review & Desk Check:

1. Individual verification of code, documents, or design before peer review.

3. Peer Reviews:

1. Conducted by colleagues during development, maintenance, or operations.
2. Aims to identify ~~errors, present alternatives, and discuss~~ solutions.

✓ **Purpose:** Improve software quality by detecting defects early and enhancing collaboration.

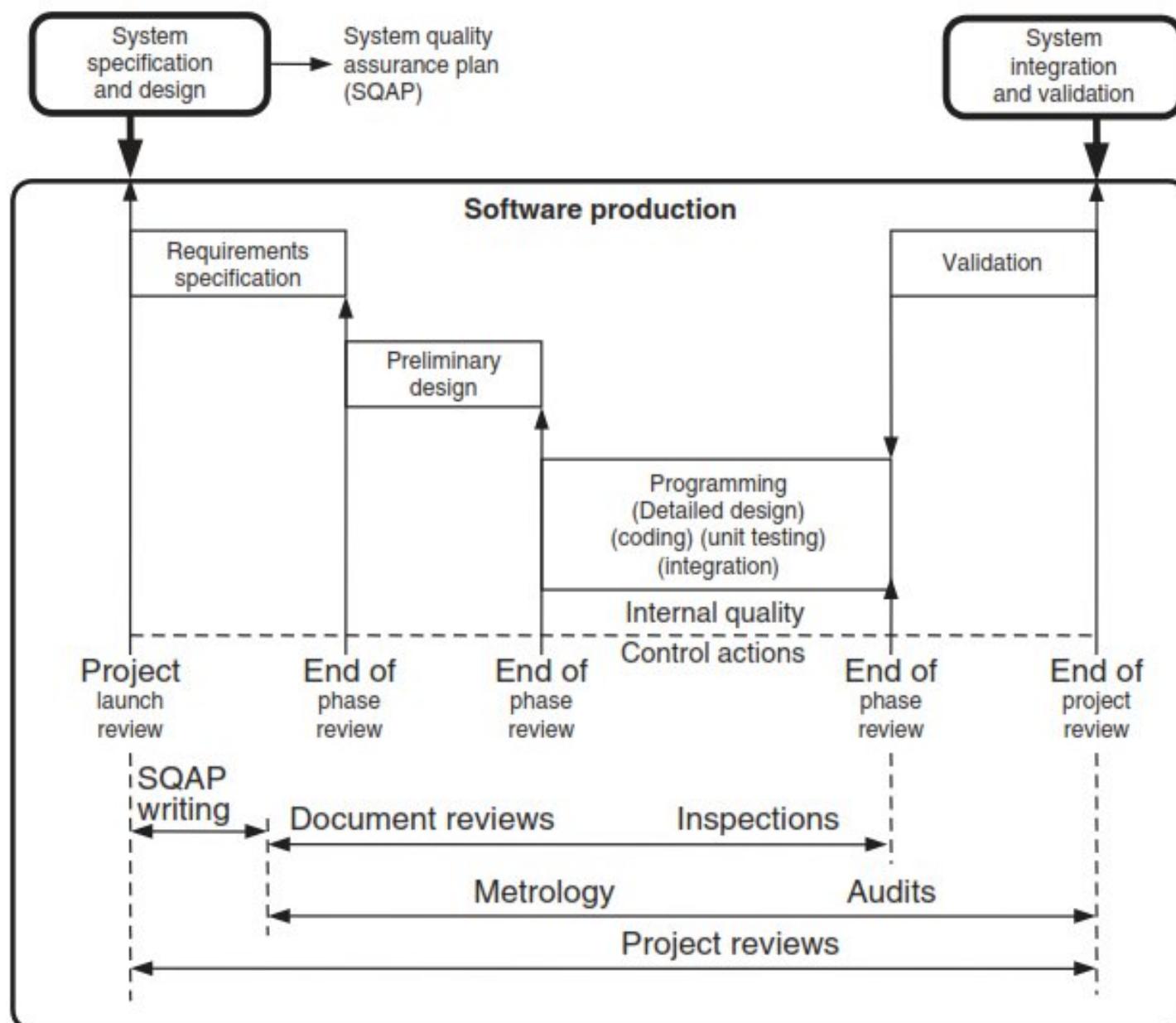


Figure 5.1 Types of reviews used during the software development cycle [CEG 90] (© 1990 – ALSTOM Transport SA).

- 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.

Sources: Adapted from Gilb (2008) [CH 08] and IEEE 1028 (IEEE 08b).

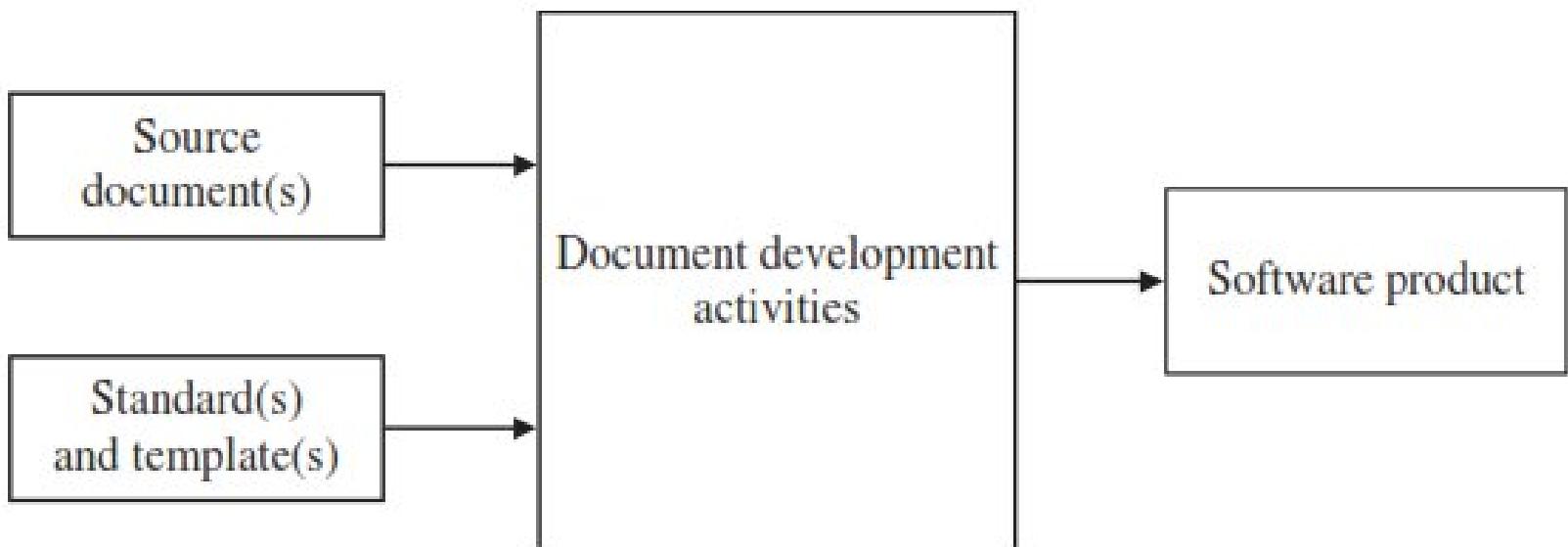


Figure 5.3 Process of developing a document.

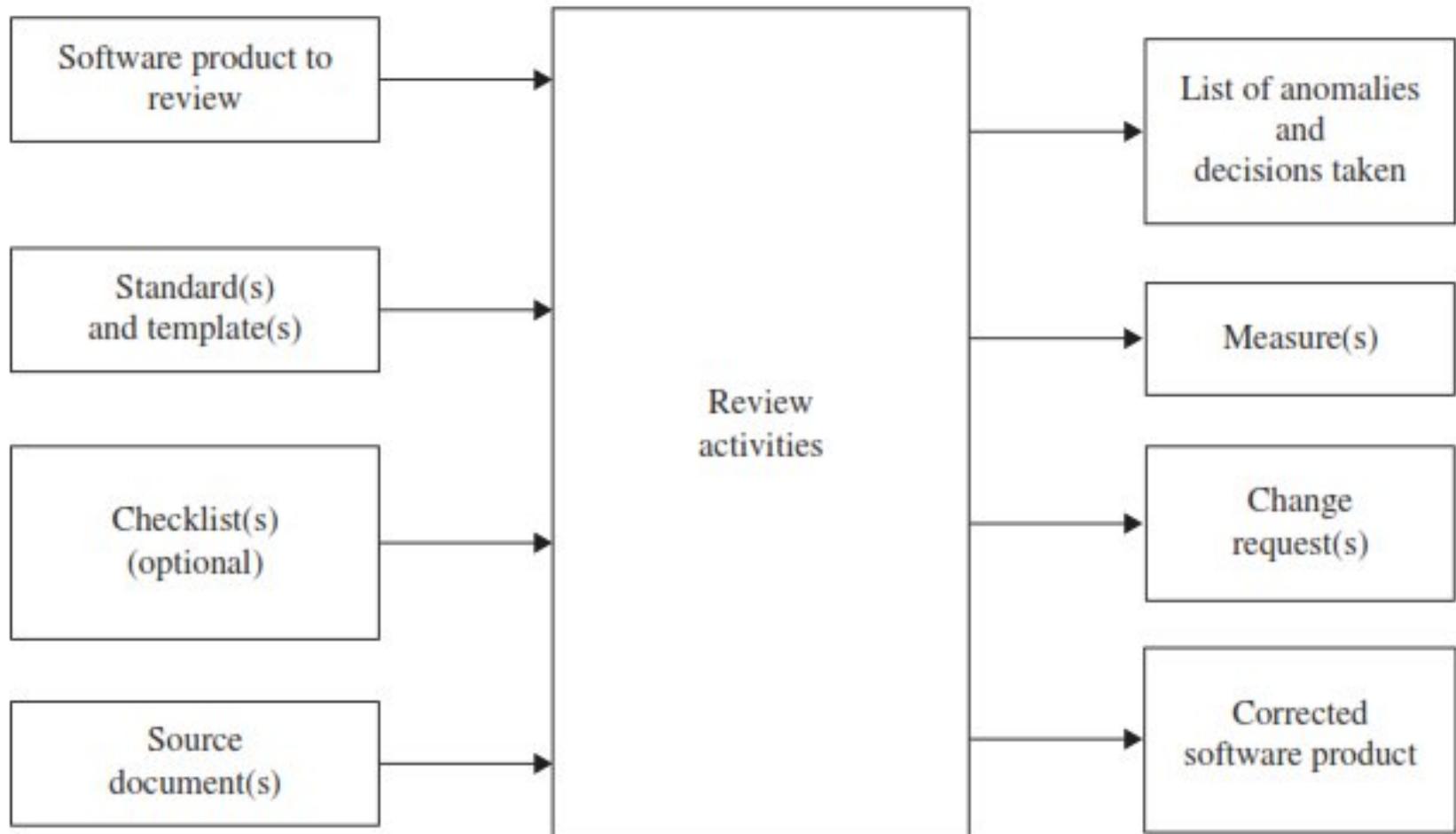


Figure 5.4 Review process.

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**.

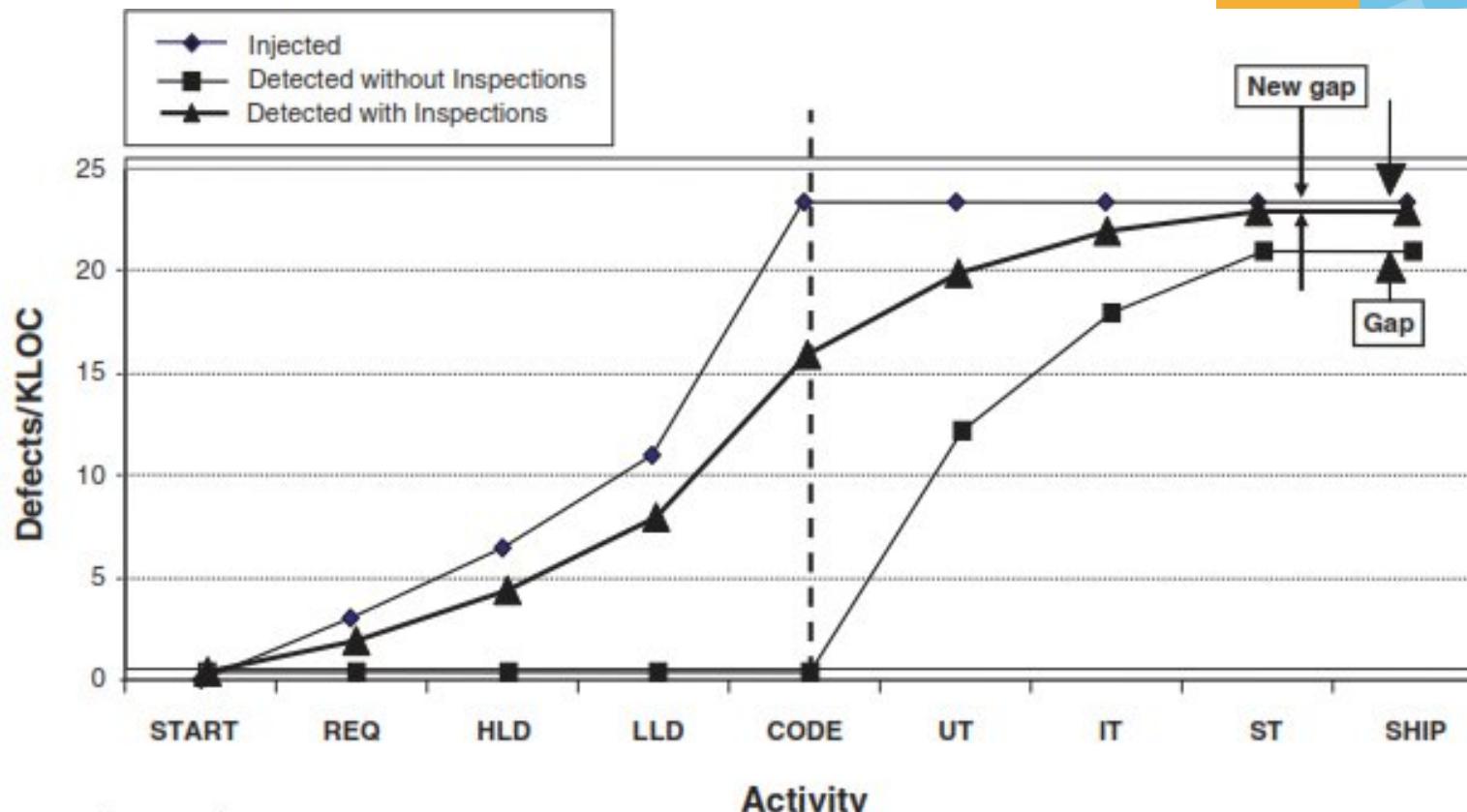


Figure 5.5 Error detection during the software development life cycle [RAD 02].

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).

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.
- 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].



THANK YOU



BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan





SE ZG501

Software Quality Assurance and

Testing

Session 7

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].

Desk-Check Reviews

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

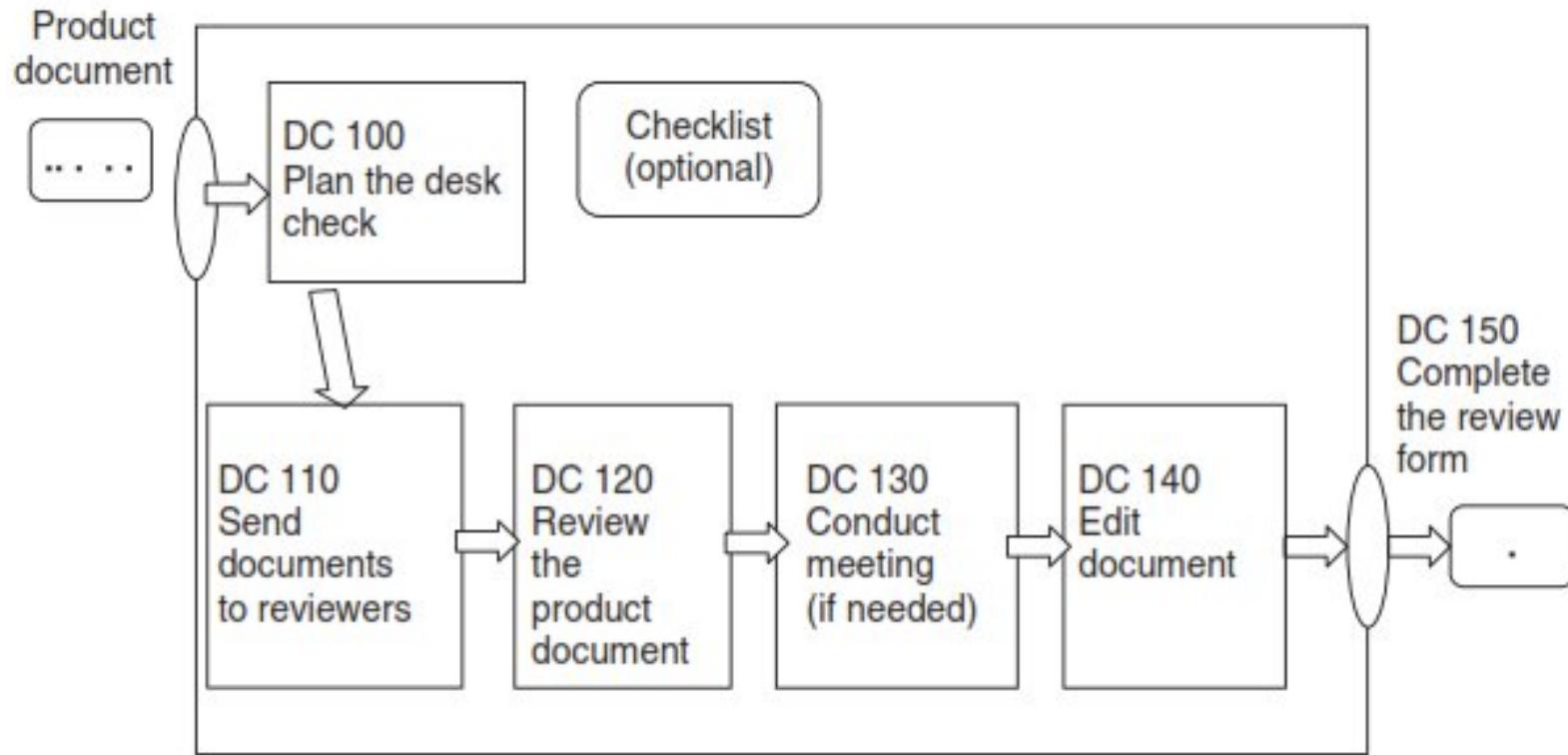


Figure 5.7 Desk-check review.

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.

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).

Figure 5.8 Desk-check review activities.

Desk check review form

Name of author: _____ Document title: _____ Document version: _____				Review date (yyyy-mm-dd): _____	
				Reviewer name: _____	
Comment No.	Document page	Line # / location	Comments	Disposition of comment* _____	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; MCD: 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.

The IEEE 1028 Standard

The **IEEE 1028-2008 Standard** defines **five types of software reviews and audits**, detailing their purpose and process.

It provides guidelines on **how to conduct systematic reviews** during **software acquisition, development, operation, and maintenance** to ensure quality and compliance.

- ✓ Defines **what to review** and **how to review it**
 - ✓ Ensures **software quality, correctness, and reliability**
 - ✓ Covers **processes for structured reviews and audits**
-

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.

IEEE 1028 -The types of reviews and audits

1. Management Review

- Evaluates project progress, schedules, and requirements.
- Conducted by management to ensure alignment with goals.

2. Technical Review

- Assesses software for functional suitability and compliance.
- Performed by technical experts to detect specification gaps.

3. Inspection

- A formal, structured review to identify errors and deviations.
- Involves step-by-step checks against standards.

4. Walkthrough

- The author presents the software to the team.
- Participants ask questions, give feedback, and suggest improvements.

5. Audit

- An independent assessment for standards and contract compliance.
- Ensures the software meets regulatory and quality requirements.

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.

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: Characteristics of Reviews and Audits (IEEE 1028 Standard)

Characteristic	Management Review	Technical Review	Inspection	Walk-through	Audit
Objective	Monitor progress	Evaluate conformance to specifications and plans	Find anomalies, verify resolution, ensure product quality	Identify anomalies, examine alternatives, improve product	Independently verify compliance with standards and regulations
Recommended Group Size	2 or more people	2 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	Responsible manager	Lead engineer	Trained facilitator	Facilitator or author	Lead auditor
Management Participation	Yes	Sometimes, if evidence is required	No	No	No, but may be called for evidence
Output	Management review documentation	Technical review documentation	Anomaly list, summary, inspection report	Anomaly list, action items, follow-up proposals	Formal audit report, findings, deficiencies

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.

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).

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

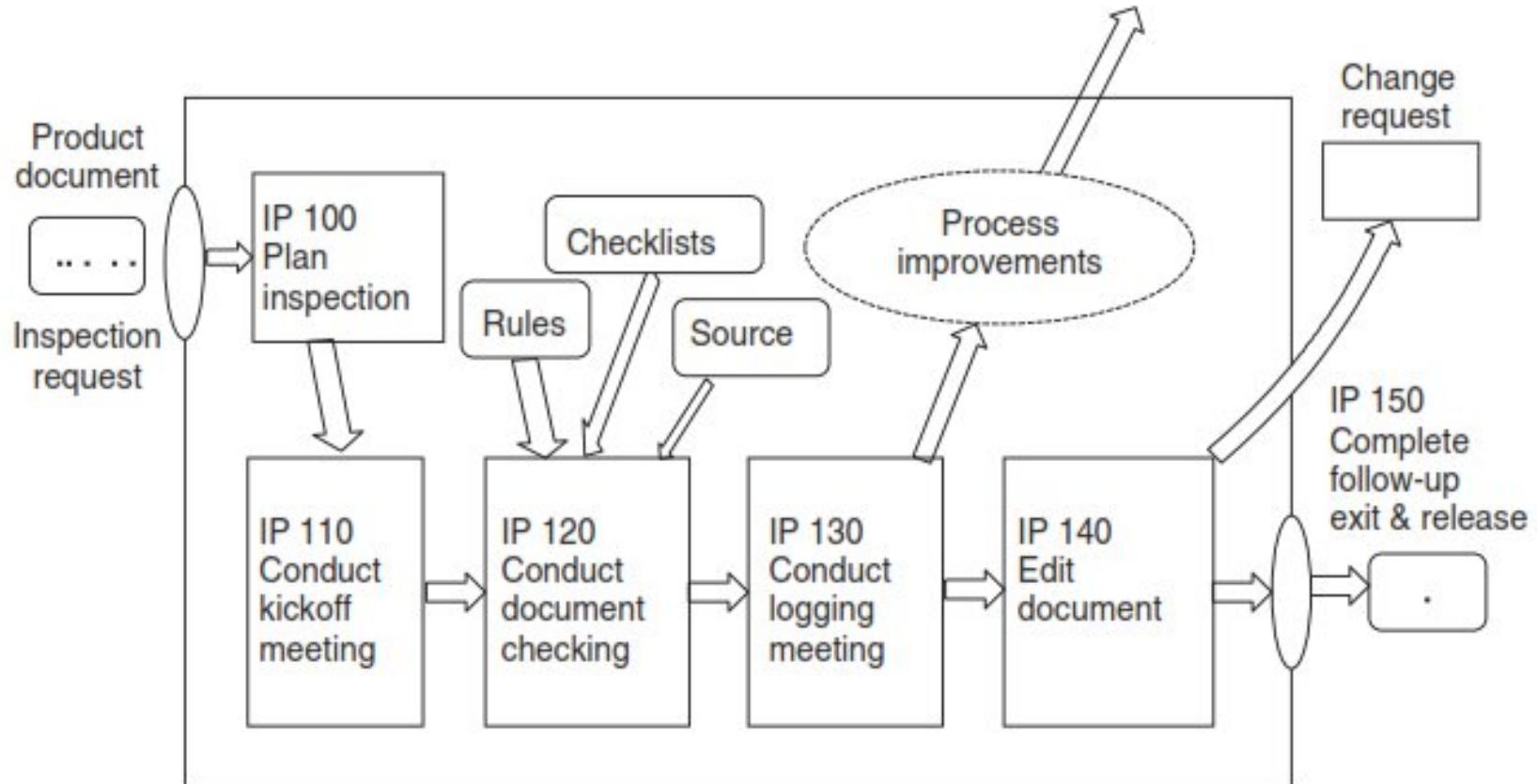


Figure 5.12 The inspection process.

PROJECT LAUNCH REVIEWS AND PROJECT 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.

QUIZ

1. What is the main purpose of a **desk-check review**?
 - A) To test software functionality.
 - B) To pass around a document for individual review and collect feedback.
 - C) To perform automated code testing.
 - D) To execute the software and detect runtime errors.

What is the main purpose of a **desk-check review**?

- A) To test software functionality.
- B) To pass around a document for individual review and collect feedback.
- C) To perform automated code testing.
- D) To execute the software and detect runtime errors.

Answer: B) To pass around a document for individual review and collect feedback

QUIZ

2. What are the **major benefits** of conducting an **inspection review**? (*Select all that apply.*)

- A) Detect and identify anomalies early in the process
- B) Improve software quality
- C) Reduce development and maintenance costs
- D) Automate software debugging

What are the **major benefits** of conducting an **inspection review?** (*Select all that apply.*)

- A) Detect and identify anomalies early in the process
- B) Improve software quality.
- C) Reduce development and maintenance costs.
- D) Automate software debugging.

Correct Answers: A, B, C

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 ?
- Who will be the team **leader**?
- What will my **role and responsibilities** ?
- 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?

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.

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

Product	Technical drivers—complexity		
	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

- **dutils** – Finds and organizes program identifiers.
- **Egrep** – Searches text using patterns.
- **Find** – Locates files on a system.
- **Diff** – Compares files to find differences.
- **Cscope** – Helps browse C code.
- **LXR** – Shows source code online with cross-referencing.

AUDITS

Audits are one of the most formal types of reviews in software quality assurance.

Different types of audits serve **different purposes**, such as:

- Ensuring a software company follows industry standards.
- Verifying that a supplier meets a client's requirements.

The **cost and independence of the auditor** depend on the type of audit being conducted.

This helps organizations **maintain compliance, improve quality, and build trust** with clients.

Key Components of a Software Audit



Management System – Defines policies, objectives, and processes for quality compliance.

Audit Criteria – Standards or policies used to evaluate compliance (e.g., ISO 9001, CMMI).

Audit Evidence – Documents, reports, and test results proving adherence to audit criteria.

Audit Process – Steps include planning, evidence collection, evaluation, reporting, and corrective actions.

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.
-

External audits

- Includes 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.

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 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.

TYPES OF AUDITS

Internal Audit

- A **first-party audit** is an internal audit conducted by an organization to assess its own processes,
- 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.

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.

- ISO does not provide certification services or issue certificates.
- Instead, certification bodies use standards like ISO 19011 for auditing guidelines and ISO/IEC 17021-1 for certification requirements.
- These standards ensure compliance with certifications like ISO 9001.

Project Assessment and Control Process



- **Align and Validate Plans:** Ensure project **plans are feasible and aligned with business objectives.**
- **Monitor Progress:** Assess the **current status of the project, technical performance, and process efficiency.**
- **Guide Execution:** Ensure the **project stays on schedule, within budget, and meets technical goals.**
- **Mandatory Reviews:** Conduct **management and technical reviews, audits, and inspections to verify compliance and performance**

CORRECTIVE ACTIONS

- After an internal or external audit, an organization must perform CAs **to correct the observed deficiencies**.
- The CA (Corrective Action) process can also be used to **address preventive actions, incident reports, and customer complaints**.
- The 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]

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**.

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

Inputs:

- Audit report
- Non-compliance issue
- Problem report

Activities:

1. Register non-conformities in the organization's issue tracking tool.
 2. Analyze and validate the problem to ensure resources are not wasted.
 3. Classify and prioritize the issue/problem based on impact.
 4. Conduct trend analysis to identify recurring problems and address them proactively.
-

Problem Resolution Process

Steps to Address the Problem:

1. Propose a Solution – Identify and suggest a fix for the problem.
2. Solve the Problem – Implement the solution and ensure it does not create new issues.
 - If the problem cannot be resolved within the project, escalate it to the appropriate management level.
3. Verify the Resolution – Check that the issue has been fully resolved.
4. Inform Stakeholders – Communicate the resolution to all relevant parties.
5. Archive Documentation – Store records of the problem and its resolution for future reference.
6. Update the Issue Tracking Tool – Ensure the tracking system reflects the latest resolution status.

Outputs:

- Resolution File – A documented report of the issue and how it was resolved.
- Corrected Software Version – An updated version of the software with the issue fixed.

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.



THANK YOU