SOFTWARE ENGINEERING
MODULE 4

SOFTWARE ENGINEERING
MODULE 4 SYLLABUS

How to implement – practices to follow for development – language, platform choices, coding practices, cost of bugs through life cycle

Quality from requirements to release and across versions: Faults and Fixes, Reliability models: Logarithmic Poisson Model

Testing mechanisms across life cycle: Functional, system integration, user testing, testing on different platforms.Testing Tools

Quality across versions and metrics for quality; Quality Models: ISO, CMM, Boehm, McCall; Metrics: Process and Product metrics - LOC, Function Points, Token Count

Key Software Development Practices : Software Process Models

Different software process models that guide development:

- **Waterfall Model** – A linear, phase-based approach.
- **Incremental Development** – Software is built and delivered in multiple increments.
- **Spiral Model** – Risk-driven iterative development.
- **Agile Development** – Flexible, customer-centric, and iterative.
- **DevOps Practices** – Focuses on continuous integration, deployment, and monitoring.

Key Software Development Practices : Requirements Engineering

A structured process to gather and document software requirements:

- **Requirements Elicitation & Analysis** – Identifying stakeholder needs.
- **Requirements Specification** – Structuring requirements clearly.
- **Requirements Validation** – Ensuring correctness and feasibility.

Key Software Development Practices : Software Design

Design principles and methodologies for effective software architecture:

- **Modularity** – Dividing the system into independent components.
- **Abstraction & Encapsulation** – Hiding implementation details.
- **Design Patterns** – Reusable solutions for common design problems.
- **Software Architecture Styles** – Layered, client-server etc.

Key Software Development Practices

- **Coding Standards** – Following best practices in programming.
- **Version Control** – Using tools like Git for managing changes.
- **Code Reviews** – Peer reviews for maintaining quality.
- **Continuous Integration & Deployment (CI/CD)** – Automating builds, testing, and releases.

Key Software Development Practices : Software Testing

Ensuring software quality and reliability:

- **Unit Testing** – Testing individual components.
- **Integration Testing** – Verifying module interactions.
- **System Testing** – Validating the complete system.
- **Acceptance Testing** – Ensuring requirements are met.

Key Software Development Practices : Risk Management

- **Identifying potential risks** early in development.
- **Analyzing and mitigating risks** through proper planning.
- **Continuous monitoring of risks** throughout the software lifecycle.

Programming Language Selection

choosing the right programming language depends on Factors for Language Choice:

- **Project Requirements** – Some domains have preferred languages (e.g., Python for AI, C for embedded systems).
- **Performance Needs** – Low-level languages like C/C++ offer better performance, while high-level languages like Python improve productivity.
- **Development Speed** – Scripting languages (Python, JavaScript) allow rapid prototyping.
- **Team Expertise** – Choosing a language that developers are skilled in improves efficiency.
- **Platform Compatibility** – Ensuring language support across required platforms.

Programming Language Selection

Common Language Choices in Software Development:

Domain	Recommended Languages
Web Development	JavaScript, Python, PHP, Ruby
Mobile Development	Swift (iOS), Kotlin (Android), React Native
System Programming	C, C++
AI & Machine Learning	Python, R
Enterprise Applications	Java, C#
Embedded Systems	C, Assembly
Cloud & DevOps	Python, Go

Key Considerations For Platform Selection:

The choice of platform impacts software architecture, performance, and user experience.

- **Target Users** – Web, mobile, desktop, or cloud-based users?
- **Scalability** – Cloud-based platforms (AWS, Azure) offer flexibility.
- **Cross-Platform Support** – Frameworks like multi-platform development.
- **Security Needs** – Some platforms provide better security measures (e.g., Linux for secure applications).

Common Platform Choices in Software Development:

Platform Type	Examples
Web-Based	AWS, Azure, Google Cloud, Node.js
Mobile	iOS (Swift), Android (Kotlin)
Cross-Platform	React Native, Flutter, Xamarin
Desktop	Windows (.NET, C#), Linux (C, Python)
Embedded Systems	Arduino, Raspberry Pi, RTOS

Coding Standards & Style Guidelines

A. Coding Standards & Style Guidelines

- **Consistent Naming Conventions** – Use meaningful names (`camelCase`, `snake_case`).
- **Code Documentation** – Add inline comments and maintain API documentation.
- **Indentation & Formatting** – Follow consistent formatting (PEP 8 for Python, Google Java Style).
- **Avoiding Hardcoded Values** – Use configuration files or constants instead.
- **Modular Programming** – Use functions and classes to improve reusability.

Coding Standards & Style Guidelines

B. Version Control & Collaboration

- **Use Git for Version Control** – Follow Git workflows (`Git Flow`, `GitHub Flow`).
- **Code Reviews** – Conduct regular peer reviews to ensure quality.
- **Branching Strategies** – Use `feature branches` and `merge requests`.

C. Error Handling & Debugging

- **Exception Handling** – Use `try-catch` blocks to prevent crashes.
- **Logging Mechanisms** – Implement logging (e.g., Log4j for Java, logging module in Python).
- **Automated Testing** – Write unit tests (JUnit for Java, PyTest for Python).

Software Testing Introduction

- software development faces many challenges.
- Software is becoming complex, but the demand for quality in software products has increased.
- software quality assurance and software testing has gained so much popularity in the last decade.
- It increases the workload and responsibility of the software development team
- If the testing team claims the presence of bugs in the software, then the development team cannot release the product.
- Testing have become essential knowledge for software developers

Evaluation Of Software Testing

The evolution of software testing was in three phases, namely

Software Testing 1.0

- software testing was just considered a single phase
- No test organization was there.
- A few testing tools were present but their use was limited due to high cost.
- Management was not concerned with testing

EVALUATION OF SOFTWARE TESTING

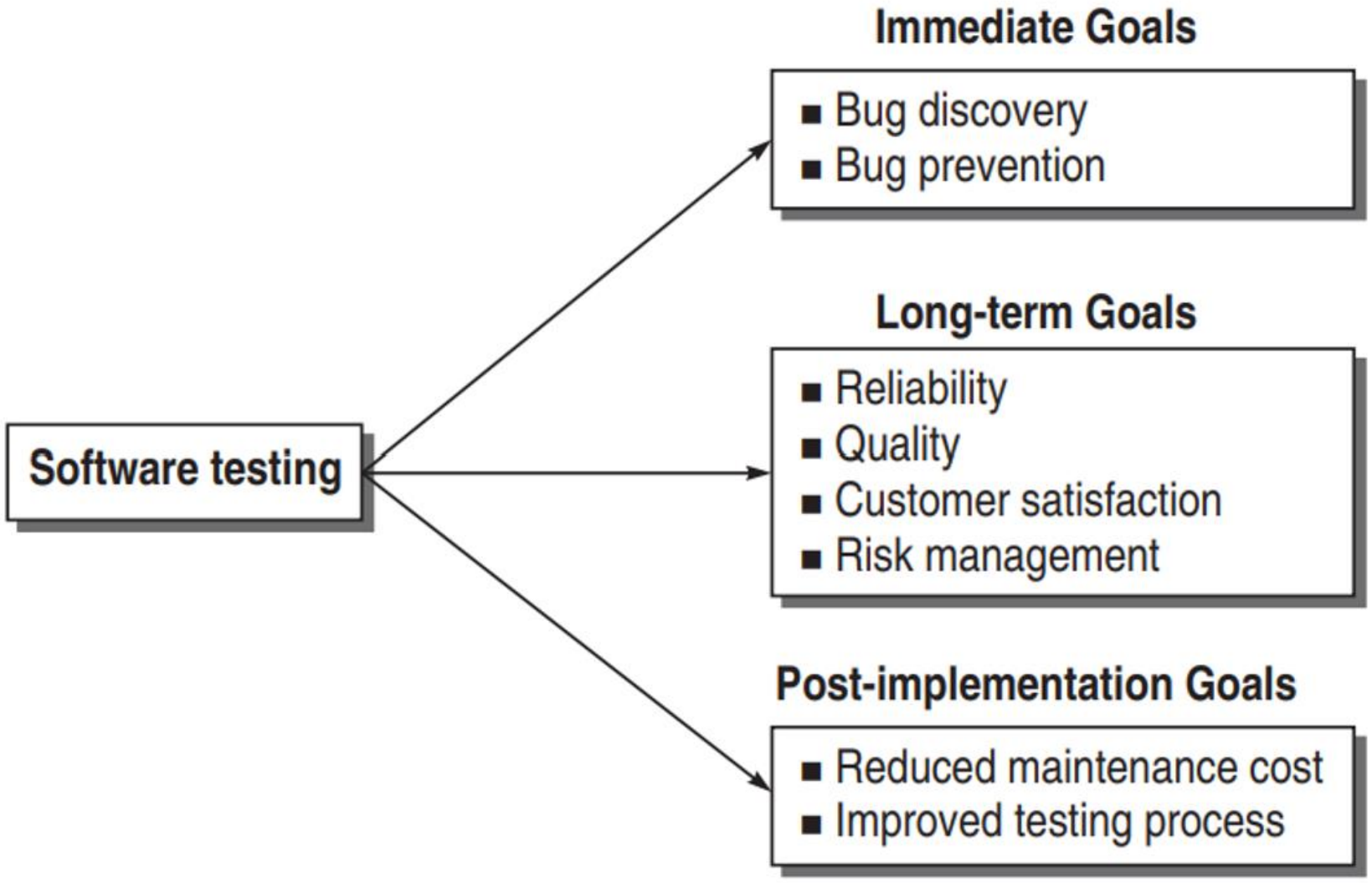
Software Testing 2.0

- concept of early testing also started.
- Testing was evolving in the direction of planning the test resources.
- Many testing tools were also available in this phase

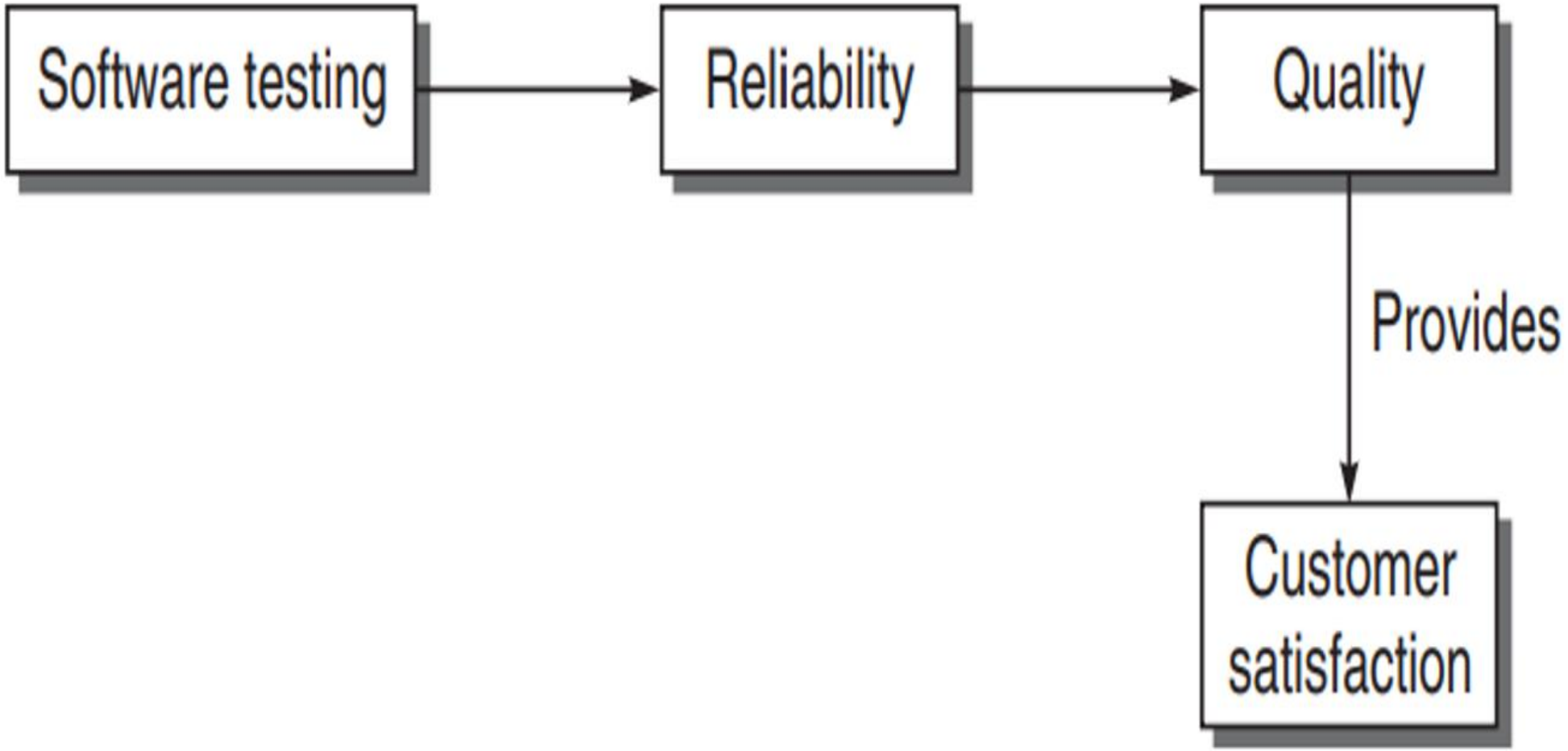
Software Testing 3.0

- It has evolved in the form of a process
- It gives a roadmap of the overall testing process.
- management is actively involved

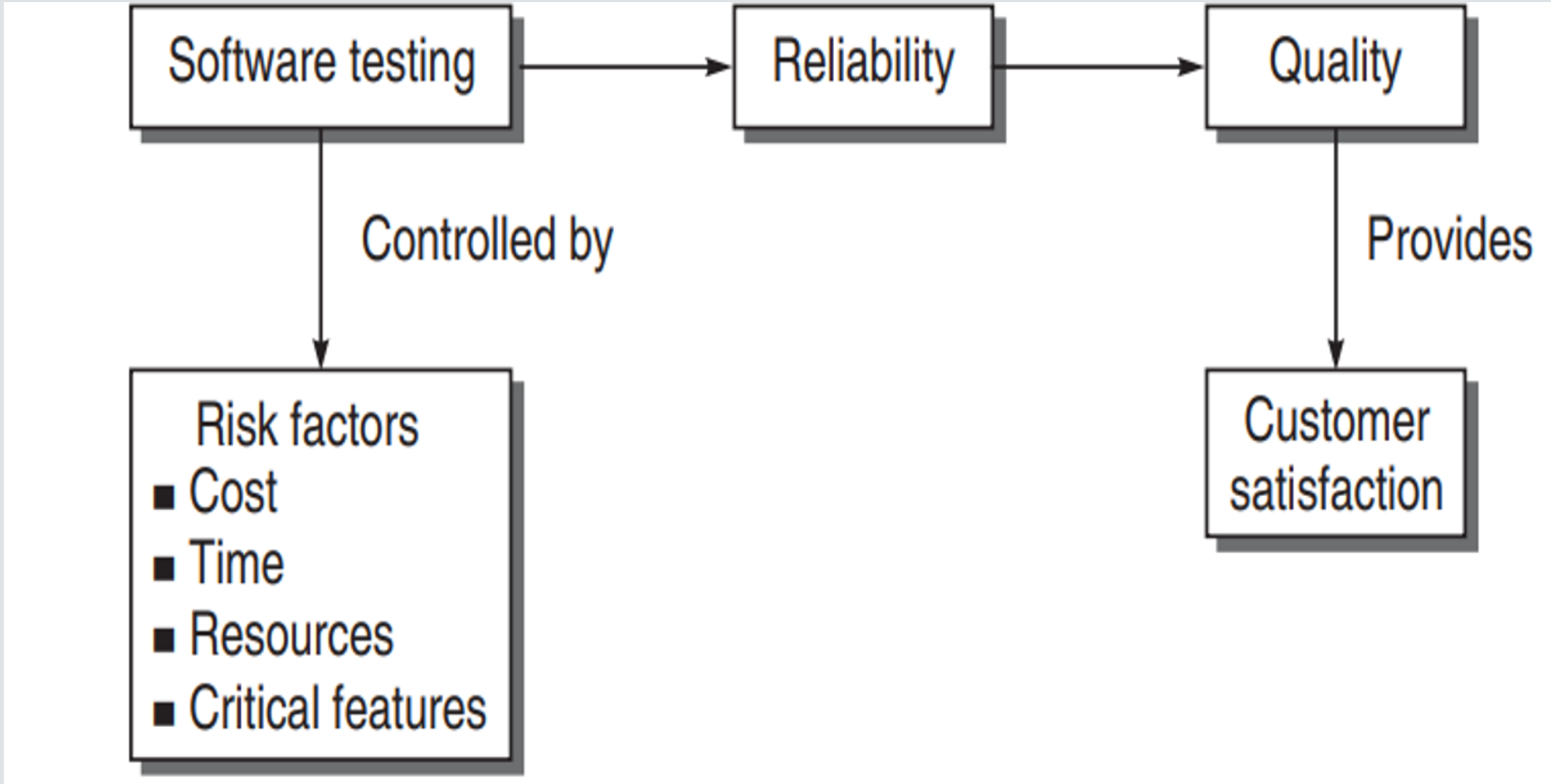
Goals Of Software Testing



Goals Of Software Testing :Customer Satisfaction



Goals of Software Testing: Risk Management

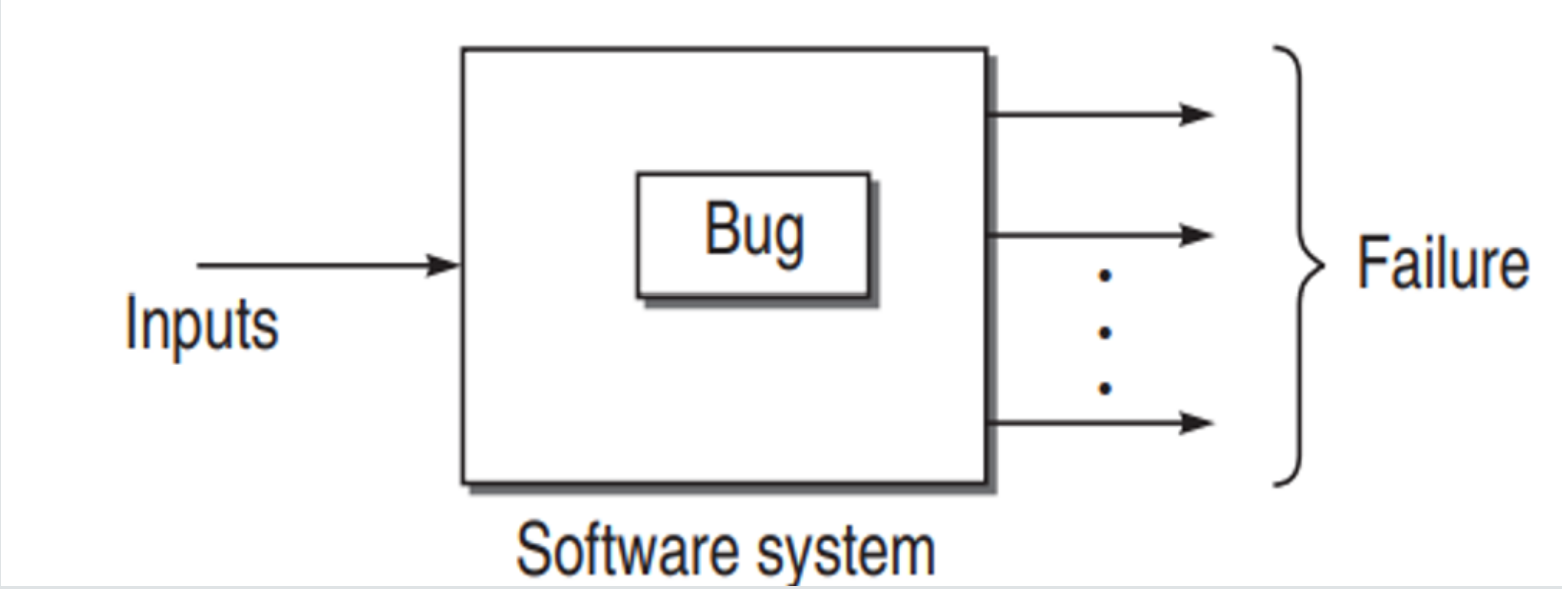


Software Testing Terminology : Failure

- It is the first term being used
- It means the inability of a system or component to perform a required function according to its specification
- when results or behaviour of the system under test are different as compared to specified expectations, then failure exists

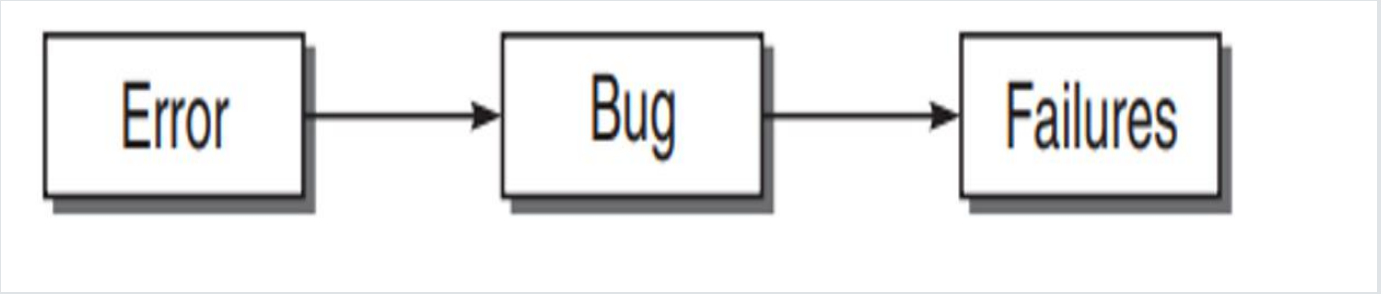
Software Testing Terminology: Fault/Defect/Bug Failure

- Fault is a condition that causes a system to produce failure.
- fault is the reason embedded in SDLC and results in failures.
- when a bug is executed, then failures are generated.



Software Testing Terminology: Error

- Error are human mistakes.
- It might be a typographical error, a misleading of a specifi cation, a misunderstanding of what a subroutine does, and so on.
- error causes a bug
- bug in turn causes failures



Software Testing Terminology: Bug Classification Based On SDLC

Bugs can appear in any phase of SDLC

Requirements and Specifications Bugs

- most of the bugs appear in this phase only.
- requirements gathered from the customer are to be converted into a requirement specification
- They may be incomplete, ambiguous, or inconsistent.
- Specification problems lead to wrong missing features

Software Testing Terminology: Bug Classification Based On SDLC

Design Bugs : Control flow bugs

- some paths may be missing; there may be unreachable paths, etc

Logic bugs Any type of logical mistakes made in the design is a logical bug.

- For example, improper layout of cases , missing cases improper combination of cases

Software Testing Terminology: Bug Classification Based On SDLC

- **Processing bugs** computation mistakes result in processing bugs.
- Examples include arithmetic error, incorrect conversion from one data representation to another, ignoring overflow
- **Data flow bugs** Control flow cannot identify data errors.
- There may be data flow anomaly errors like un-initialized data, initialized in wrong format, data initialized but not used, redefined without any intermediate use, etc.

Software Testing Terminology: Bug Classification Based On SDLC

Error handling bugs

- exception handling mechanisms must be adopted.
- If the system fails, then system should handle the error in an appropriate way.

Boundary-related bugs program goes beyond its minimum and maximum values

User interface bugs If the user does not feel good while using the software, then there are user interface bugs.

Examples misleading, or confusing information; wrong content in the help text; inappropriate error messages, etc.

Software Testing Terminology: Bug Classification Based On SDLC

Coding Bugs Programmer has long list of coding bugs

For example, undeclared data, undeclared routines

Interface and Integration Bugs External interface bugs include invalid timing or sequence assumptions related to external signals, misunderstanding external input and output formats, and user interface bugs.

System Bugs system as a whole based on various parameters like performance, stress

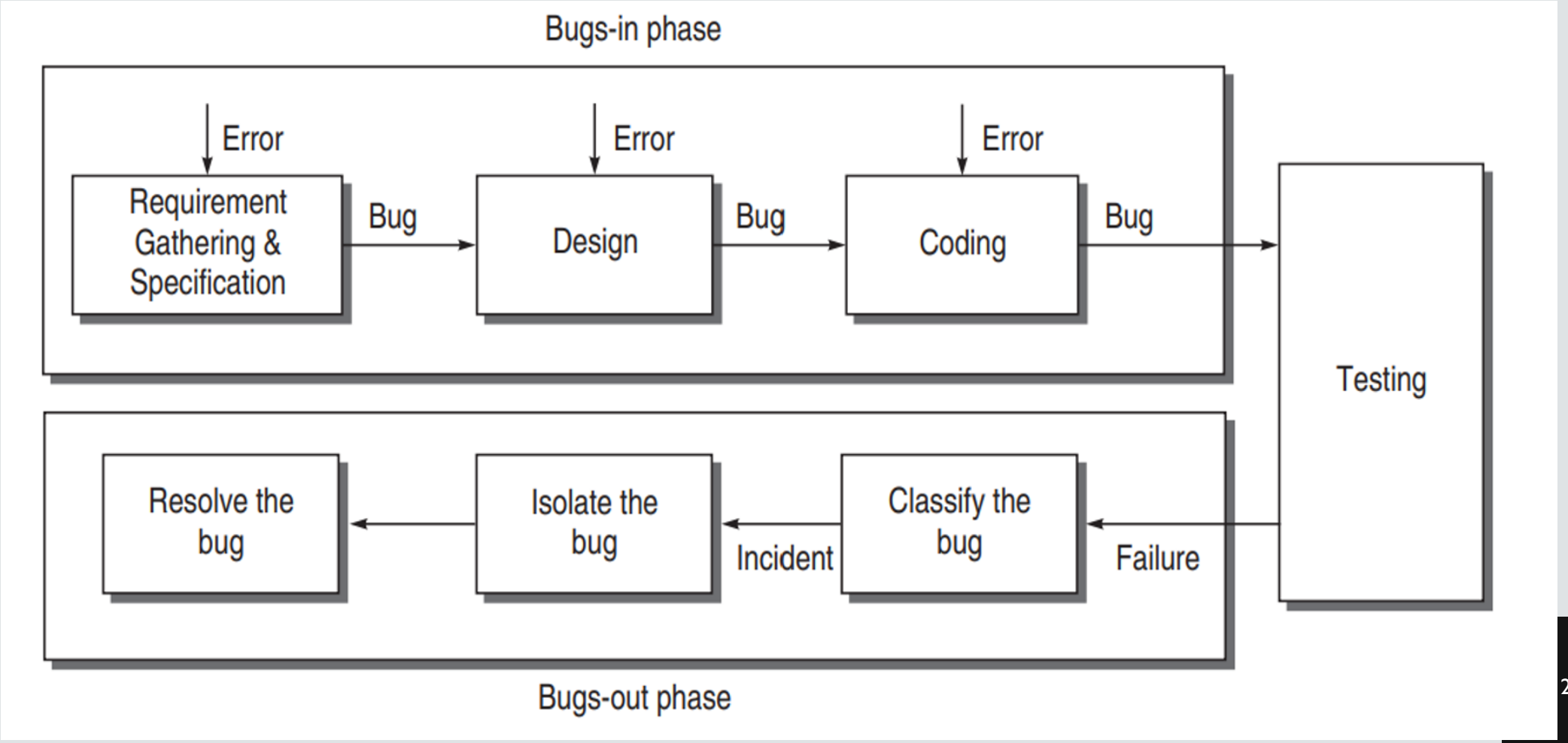
Testing Bugs testing is also performed by testers – humans.

- Some testing mistakes are: failure to notice/report a problem, failure to use the most promising test case

Software Testing Terminology: Life Cycle Of A Bug

- The whole life cycle of a bug can be classified into two phases:
- (i) bugs-in phase and (ii) bugs-out phase.
- **Bugs-In Phase**
- Whenever we commit a mistake
- it creates errors on a specific location of the software
- when error goes unnoticed
- It causes some conditions to fail, leading to a bug in the software.
- Thus, verification is required

Software Testing Terminology: Life Cycle Of A Bug



Software Testing Terminology: Life Cycle Of A Bug

Bugs-Out Phase

- there are situations when bugs are present, even though we don't observe any failures.
- In this phase, we observe failures
- we classify the bugs according to its nature.
- A bug can be critical or catastrophic in nature
- But a tester may not have sufficient time.
- Thus, bugs categorization of bugs help in handling highly critical bugs

Software Testing Terminology: Life Cycle Of A Bug

Bug classification

- classify the bugs according to its nature.
- A bug can be critical to catastrophic in nature

Bug isolation it is the activity by which we locate the module in which the bug appear

- We observe the symptoms and back-trace the software and reach the module this known as bug isolation.

Bug resolution Once we pinpoint the location of the bug.

- bug is resolved when we have found the exact location of its occurrence.

Software Testing Terminology: States Of A Bug

bug attains the following different states in its life cycle

New The state is new when the bug is reported first time by a tester

Open the test leader approves that the bug is genuine, its state becomes open.

Assign An open bug is verified and validated by the development team

- The developer is assigned the job to fix it the state of the bug now is 'ASSIGN'.

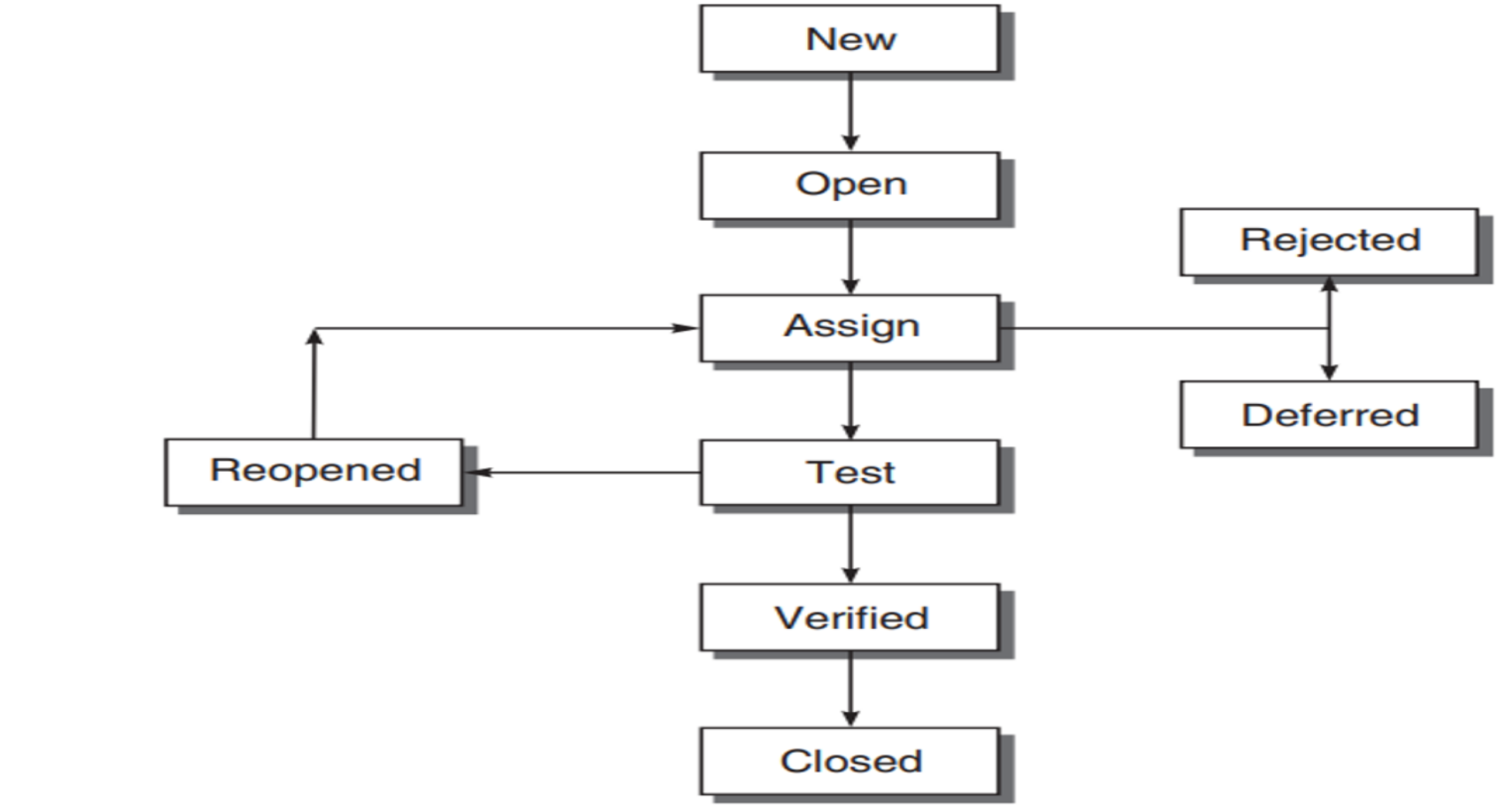
Deferred The developer Checks bug priority

- Bug adverse affects are checked
- No sufficient time to rectify bug it is expected to be fixed in next releases.

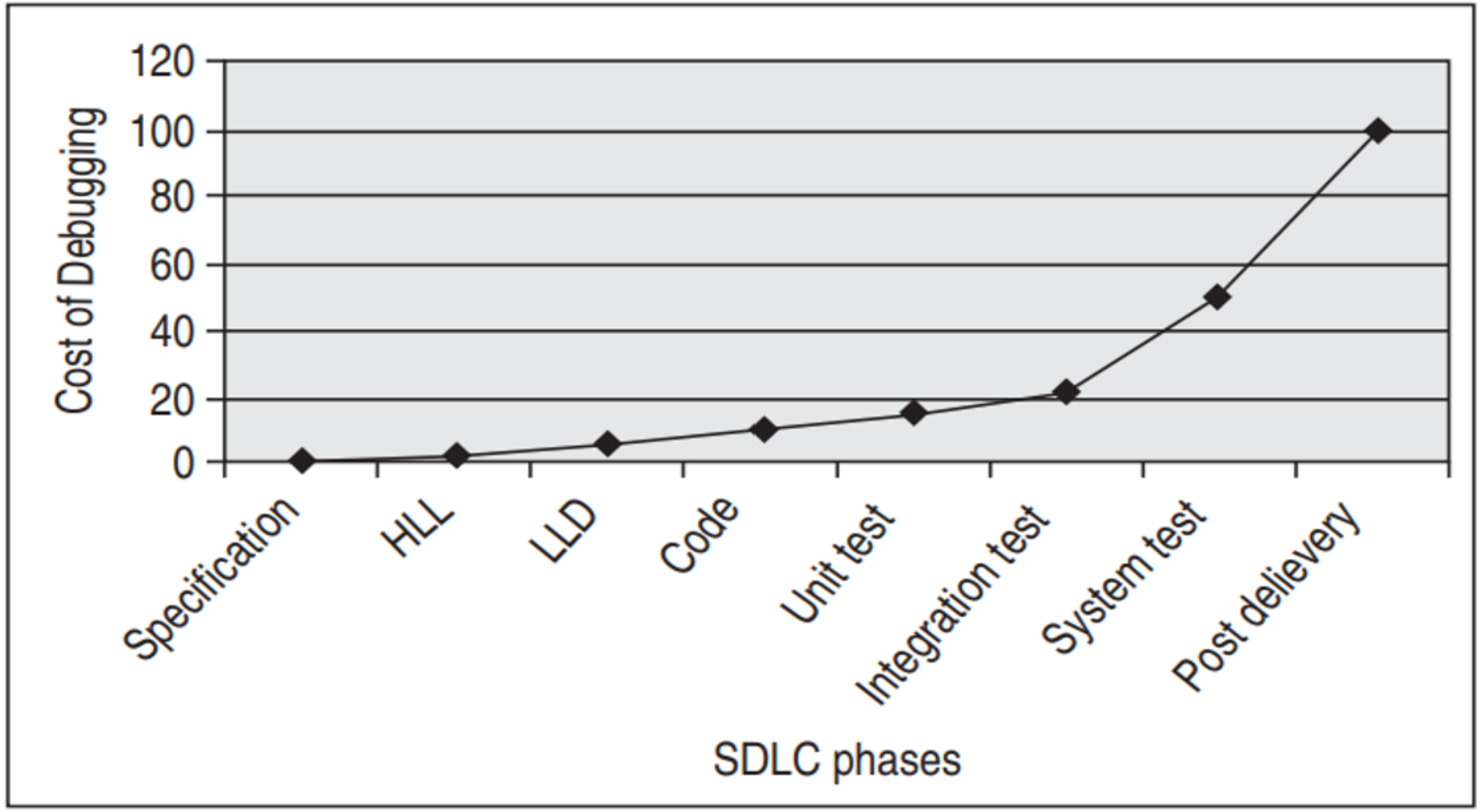
Software Testing Terminology: States Of A Bug

- **'TEST'.** After fixing the valid bug
- the developer sends it back to the testing team for next round of checking.
- **Verified/fixed** the developer approves that the bug is fixed and changes the statu to 'VERIFIED'.
- **Reopened** If the bug is still there even after fixing it, the tester changes its status t 'REOPENED'.

Software Testing Terminology: States Of A Bug



Software Testing Terminology: Bugs Affect Economics Of Software Testing



Software Testing Terminology: Bug Classification Based On Criticality

- Bugs can be classified based on the impact they have on the software under test.
- Bug classification can be used for the prioritization of bugs
- prioritization will put high criticality bugs on top of the list.
- **Critical Bugs** it has the worst effect such as it stops or hangs the normal functioning of the software.
- The person using the software becomes helpless when this type of bug appears
- the system hangs and needs to be reset.

Software Testing Terminology: Bug Classification Based On Criticality

- **Major Bug** it does not stop the functioning of the software
- it causes a functionality to fail to meet its requirements as expected.

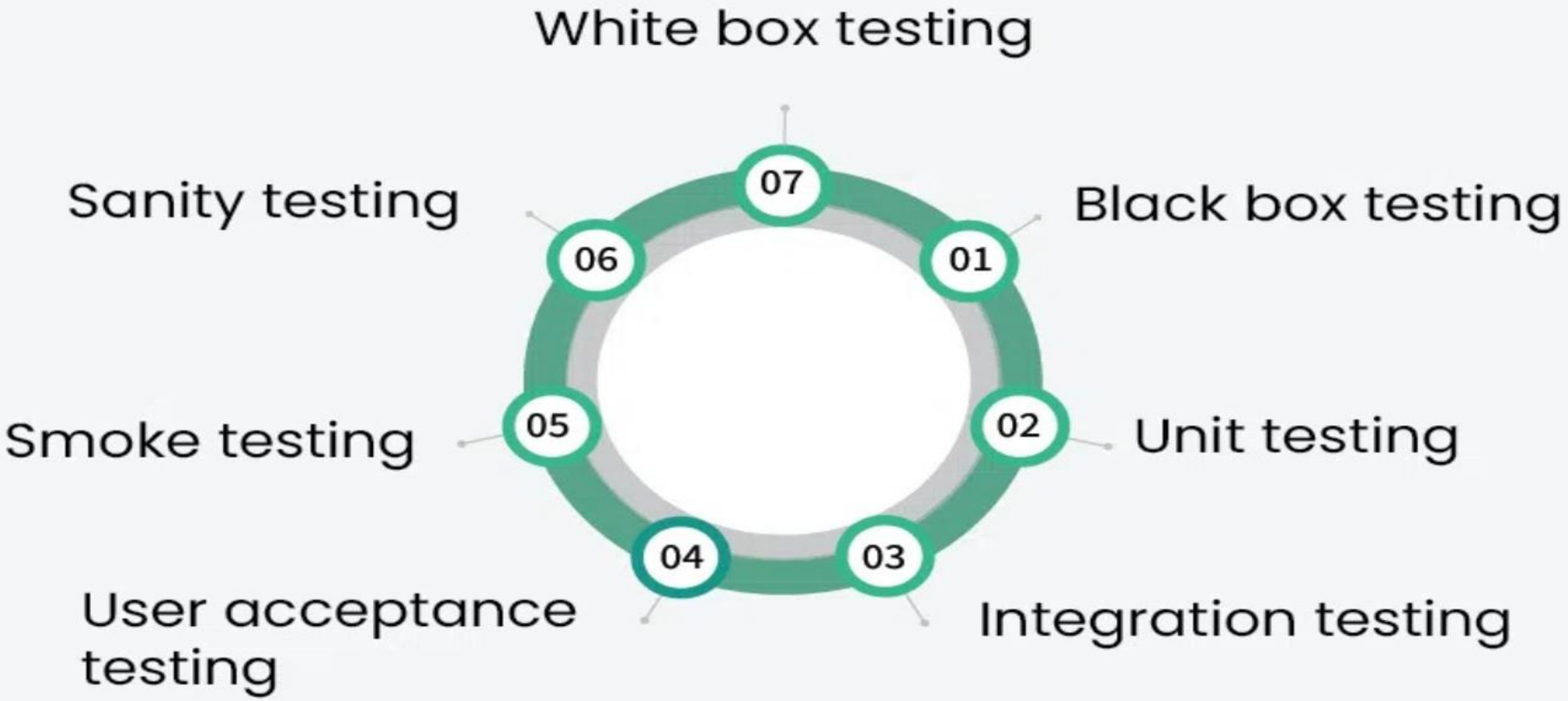
Medium Bugs they are less critical

- If the outputs are not according to the standards or conventions
- e.g. redundant or truncated output, then the bug is a medium bug.

Minor Bugs These are just mild bugs

- They occur without any effect on the expected behaviour or continuity of the software.
- For example, typographical error or misaligned printout.

Functional testing



FUNCTION TESTING

- functionality of the system specification is tested
- function coverage metric is used To keep a record of function testing
- It detect discrepancies between the functional specifications of a software and its actual behaviour.
- It verify that the system behaviour

BLACK BOX TESTING

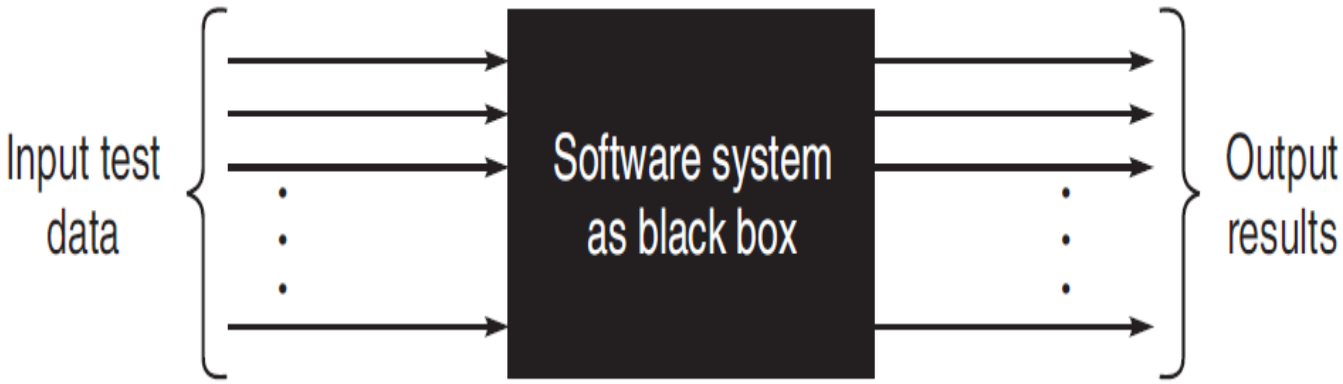


Figure 4.1 Black-box testing

BLACK BOX TESTING

- **Blackbox tests** Blackbox tests **do not deal with the internal aspects** of the component, no with the behavior or the structure of the components.
- Test cases are designed based on functional specifications.
- Input test data are checked against expected outputs
- It look for interface errors
- test the system behaviour and check its performance
- test the maximum load or stress on the system
- **Whitebox tests** focus on the internal structure of the component.

FUNCTION TESTING

Table 7.4 Function coverage matrix

Functions/Features	Priority	Test Cases
F1	3	T2,T4,T6
F2	1	T1, T3,T5

ACCEPTANCE TESTING: FORMAL TESTING

- Determine whether a software system satisfies its acceptance criteria to accept the system or not.
- software built satisfy the user requirements
- customer/client must be involved
- end user provide the development team with feedback
- It is carried out by end-users.
- Determine whether the software is fi t for the user.
- Making users confident about the product.
- Check if software system satisfi es its acceptance criteria.

TYPES OF ACCEPTANCE TESTING

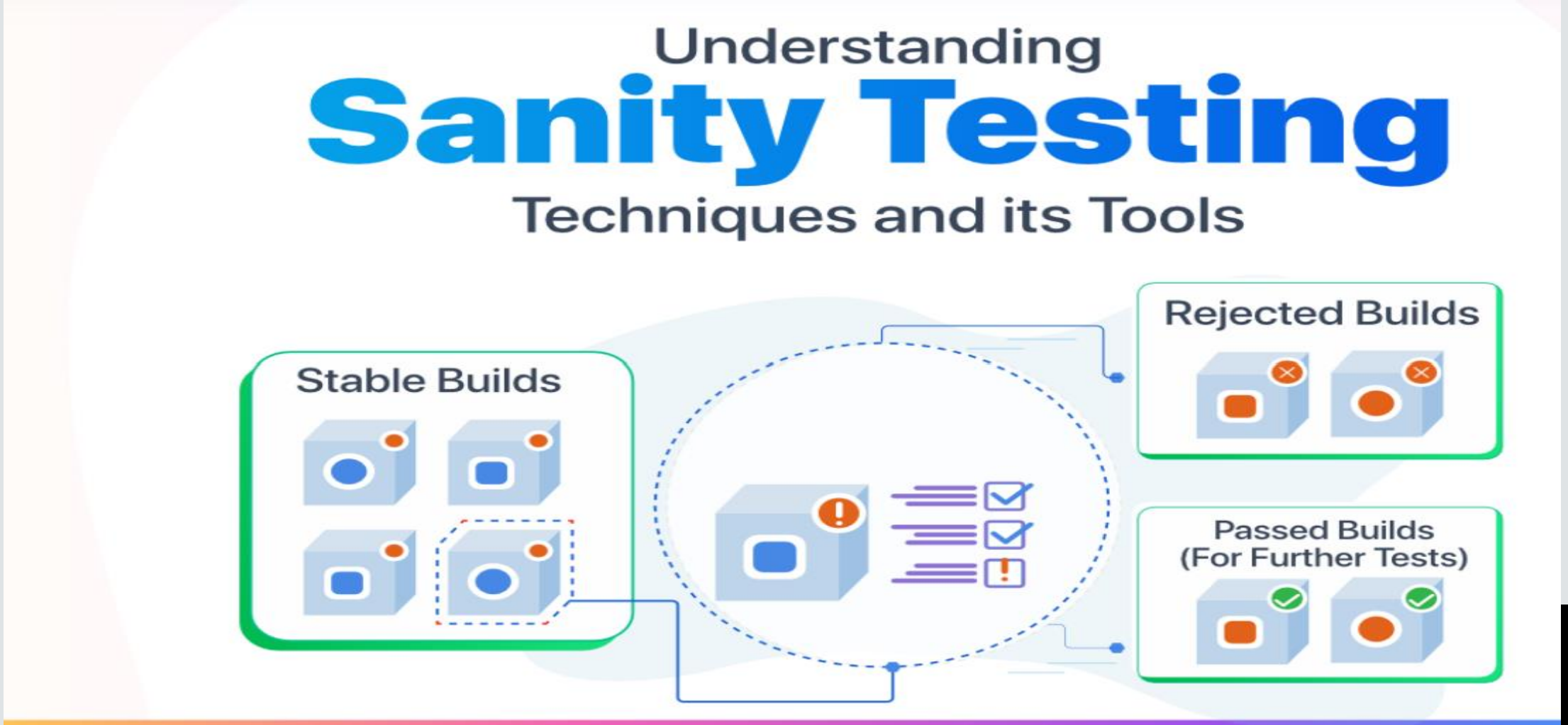
Acceptance testing is classifi ed into two categories:

- Alpha Testing Tests are conducted at the development site by the end users.
- Beta Testing Tests are conducted at the customer site

SANITY TESTING

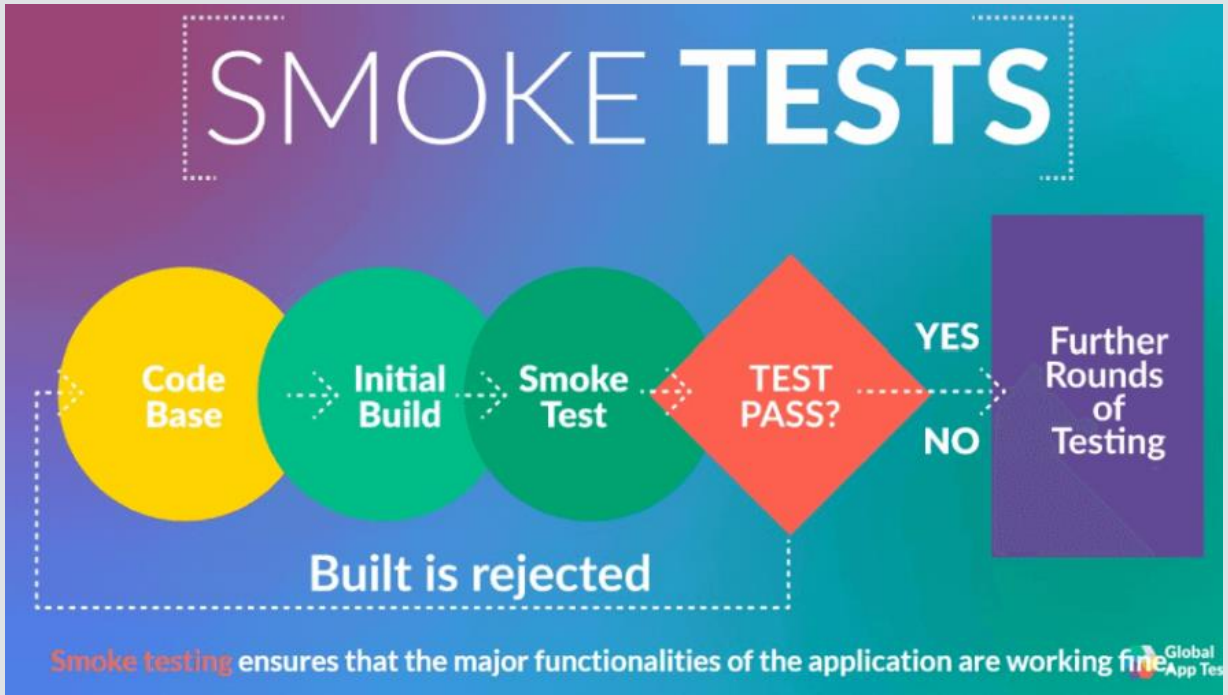
- It verifies that minor changes to a software application have not introduced new issues.
- What is it used for?
- To ensure that the core functionality of the software remains intact
- To verify that the build is stable enough for further testing
- check whether the application is built according to user needs

SANITY TESTING

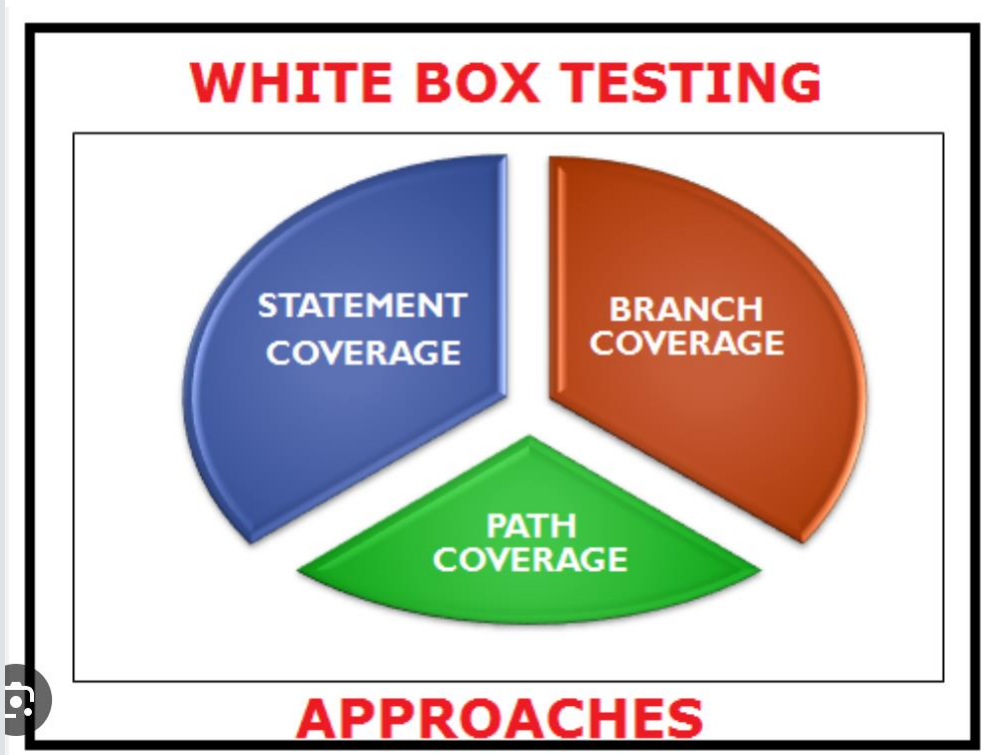


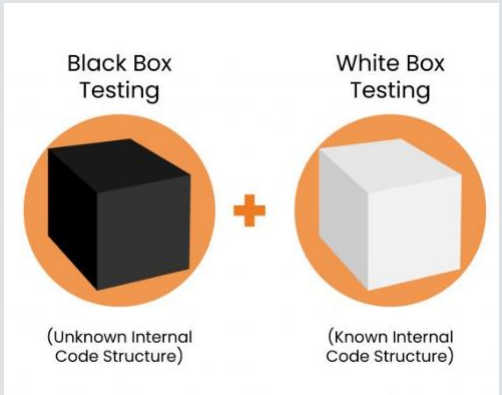
SMOKE TESTING

- Smoke testing is a preliminary testing process that checks if the critical features of an application work properly before proceeding with more detailed testing.
- Identifies critical issues that could prevent further testing

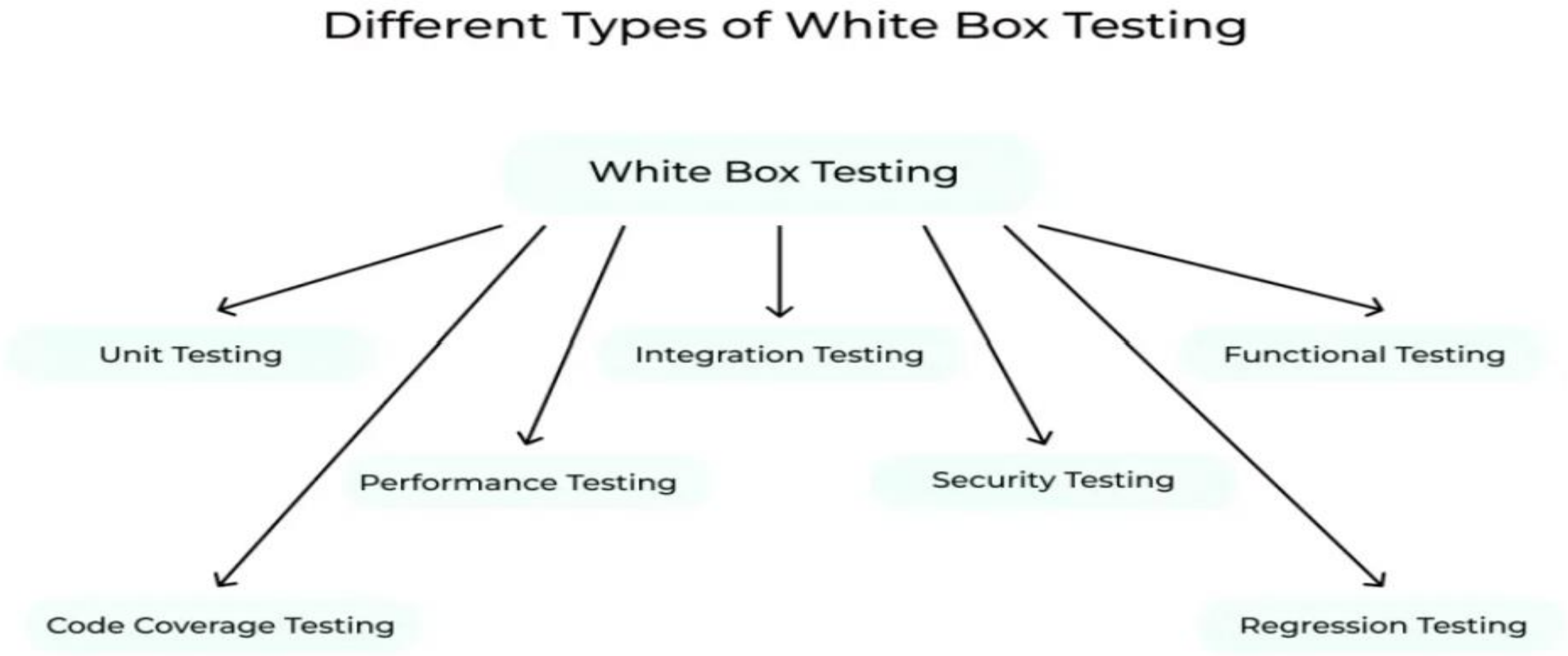


WHITE BOX TESTING





WHITE BOX TESTING



Unit Testing

- Three levels of validation testing.

Unit Testing

- major validation effort performed on the smallest module of the system.
- If avoided, many bugs become latent bugs
- basic level of testing
- It confirms the behaviour of a single module according to its functional specifications.

Integration Testing

- It combines all unit-tested modules and performs a test on their aggregation
- unit modules are not independent, and are related to each other by interface specifi cations
- Interfacing between units must be tested.
- It is actually a systematic technique for combining modules.
- We integrate the units according to the design and availability of units.
- Tester must be aware of the system design.

Integration Testing

- It exposes inconsistency between the modules such as
- Data can be lost across an interface.
- module Integration may not give the desired result.
- Data types and their valid ranges may mismatch between the modules.

Integration Testing

- Nodes represent the modules present in the system
- Links/edges between the two modules
- The nodes on the last level in the tree are *leaf nodes*
- integrate all the modules together and then test it. (Non Incremental)
- Another method is to integrate the modules one by one and test them incrementally.
- Based on these methods, integration testing methods are classified into two categories:
- (a) non-incremental
- (b) incremental.

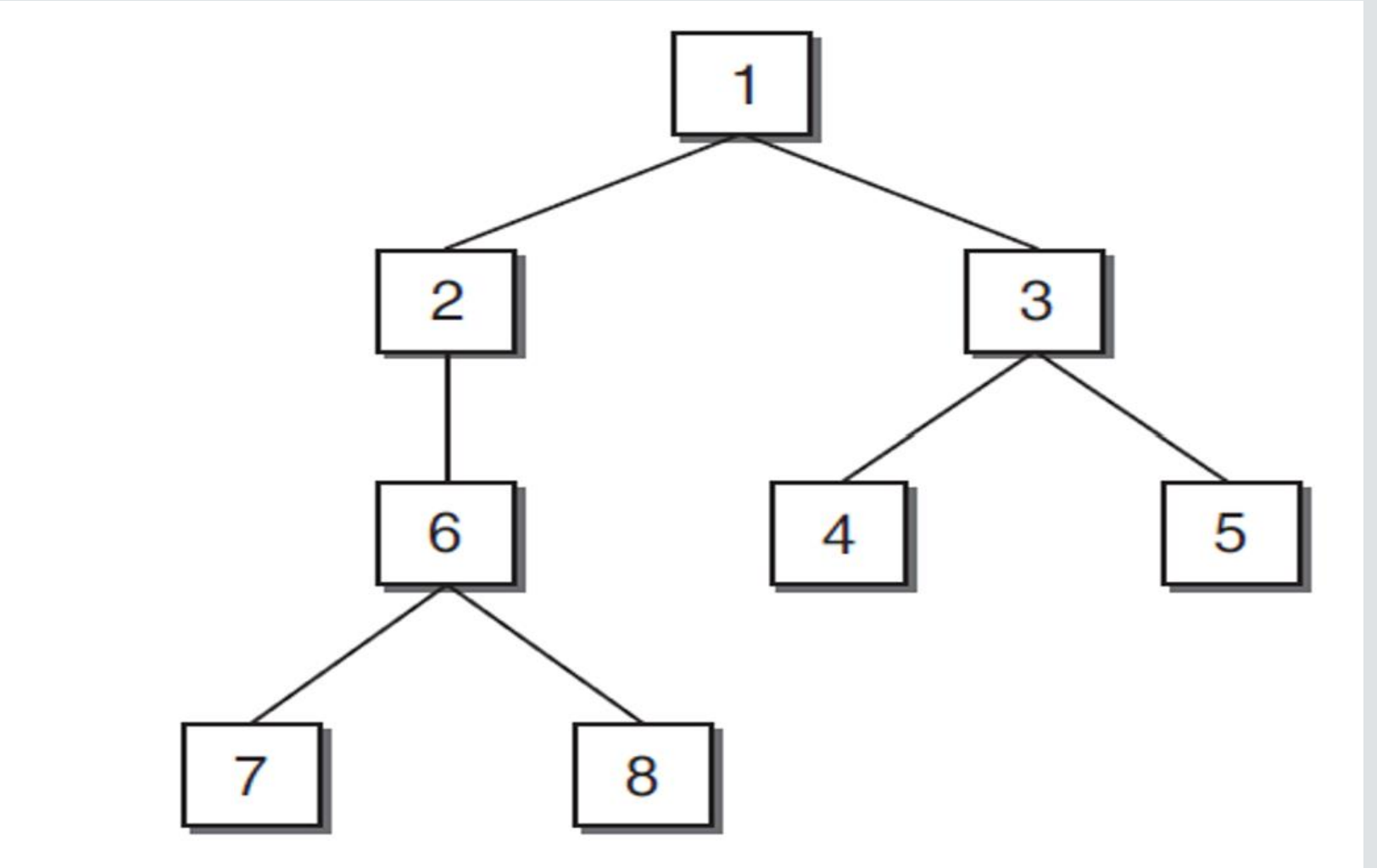
Non-incremental Integration Testing : Big-bang Integration Testing

- untested modules are combined together and then tested are combined together. It is also known as Big-Bang integration testing

Drawbacks

- Big-Bang method cannot be adopted practically
- Big-Bang requires more work.

Non-incremental Integration Testing : Big-bang Integration Testing



Incremental Integration Testing

- Start with one module and unit test it.
- Then combine with another module
- modules are combined one by one
- perform test on both the modules.
- incrementally keep on adding the modules
- It test the recent environment.
- Thus, an integrated tested software system is achieved

Types Of Incremental Integration Testing

Incremental Integration Testing Is Divided Into Two Categories.

- 1. Top-down Integration Testing
- A. *Depth First Integration*
- *B Breadth First Integration*
- 2. Bottom-up Integration Testing

Top-down Integration Testing

- Start with the top or initial module in the software.
- Substitute the stubs for all the subordinate modules of top module. Test the top module.
- After testing the top module, stubs are replaced one at a time with the actual modules for integration.
- Perform testing on this recent integrated environment.
- Look at the design hierarchy from top to bottom.
- Start with the high-level modules and move downward through the design hierarchy

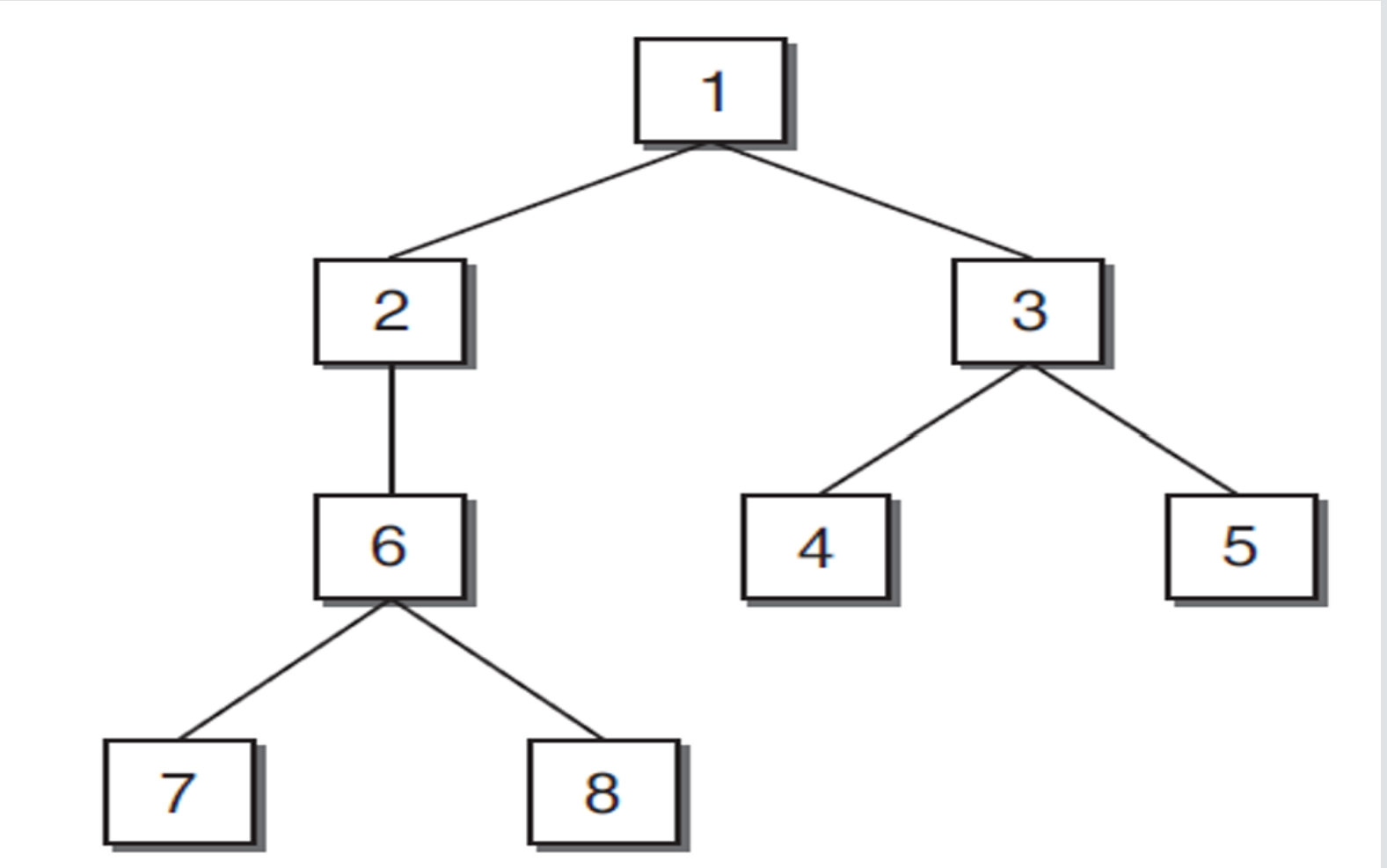
Top-down Integration Testing : Depth First Integration

- Modules subordinate to the top module are integrated in the following two ways:

- ***Depth fi rst integration*** *1* 2, 6, 7/8 will be integrated fi rst.

  Next, modules 1, 3, 4/5 will be integrated

Top-down Integration Testing : Depth First Integration

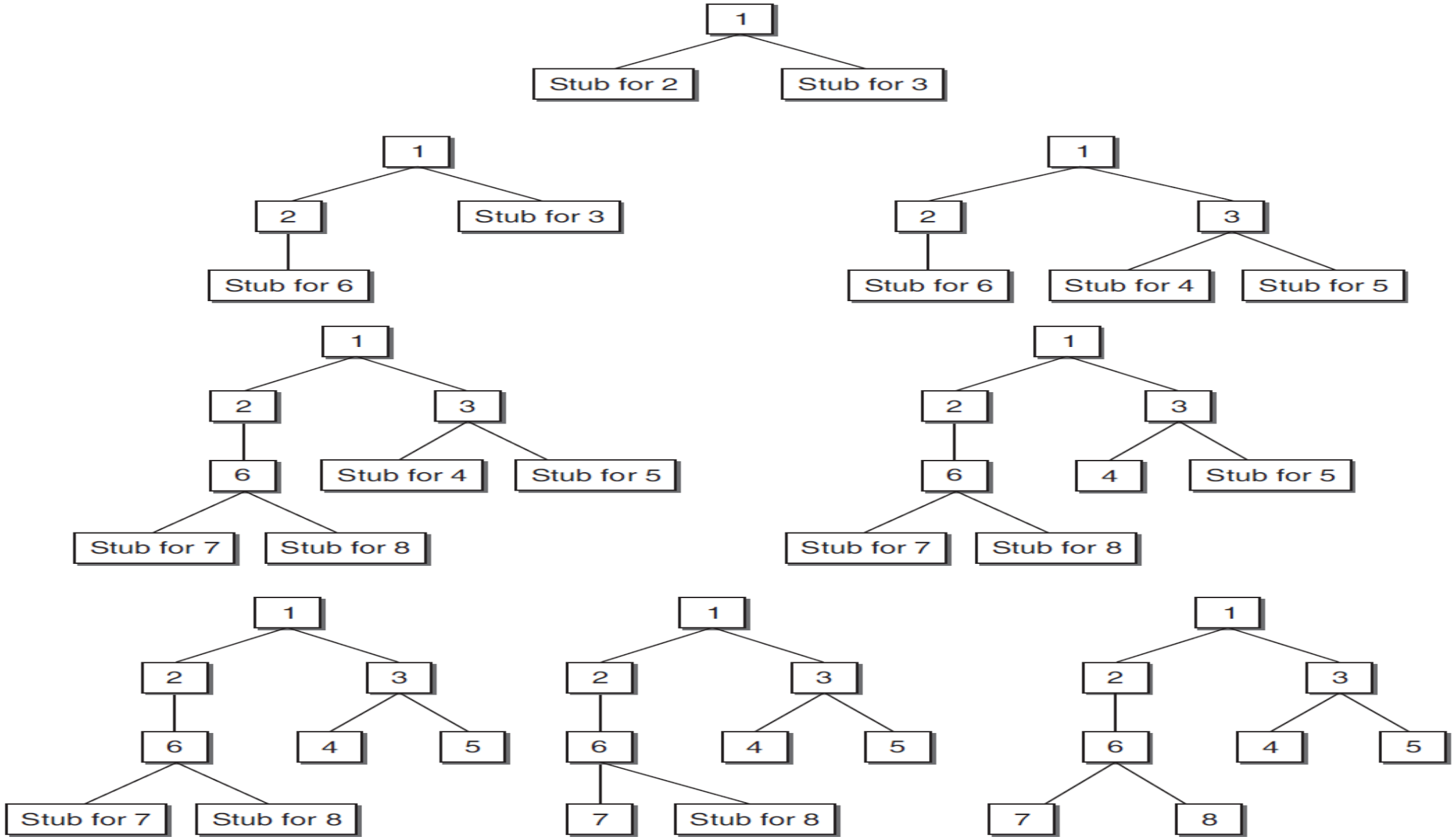


Top-down Integration Testing : Breadth First Integration

- All modules directly subordinate at each level, moving across the design hierarchy horizontally, are integrated fi rst

- modules 2 and 3 will be integrated fi rst. Next, modules 6, 4, and 5 will be integrated. Modules 7 and 8 will be integrated last.

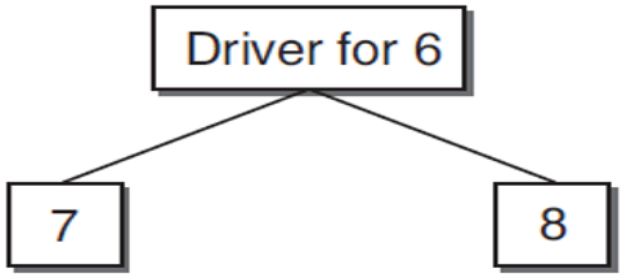
Top-down Integration Testing : Breadth First Integration

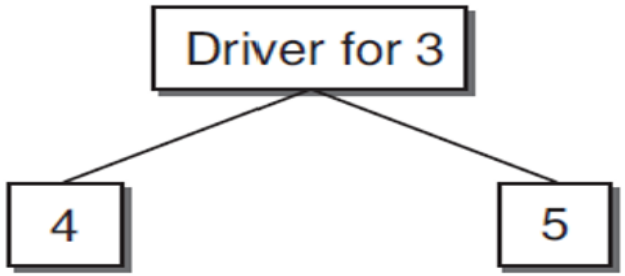


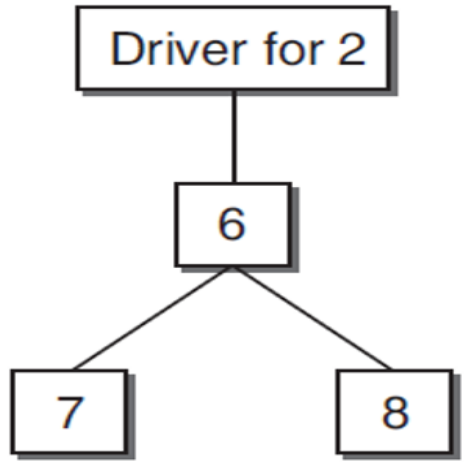
Bottom-up Integration Testing

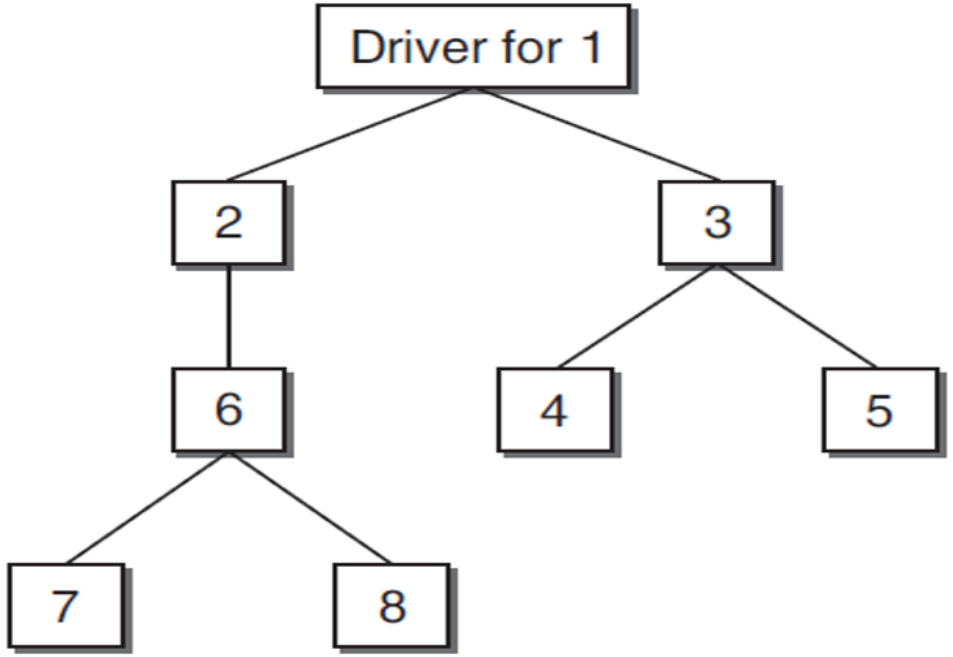
- Begins with the modules at the lowest level in the software structure.
- After testing these modules, they are integrated and tested moving from bottom to top level
- Bottom-up integration can be considered as the opposite of top-down approach
- Start with the lowest level modules in the design hierarchy.
- Look for the super-ordinate
- Design the driver module for this super-ordinate module.
- Test the module selected in step 1 with the driver designed in step 2.
- Repeat steps and move up in the design hierarchy.
- Whenever, the actual modules are available, replace stubs and drivers with the actual one and test again.

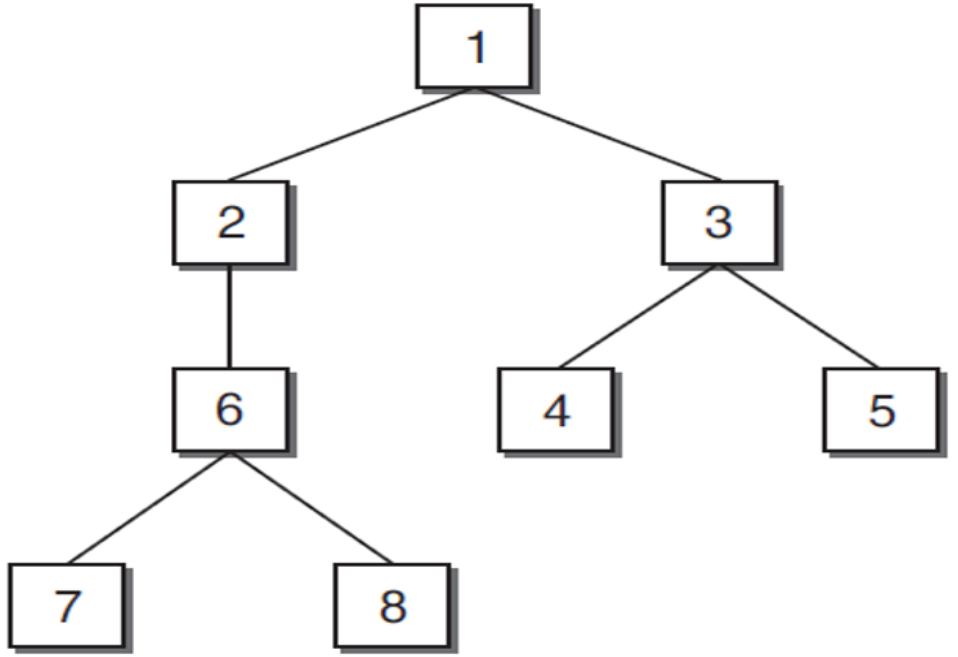
Bottom-up Integration Testing







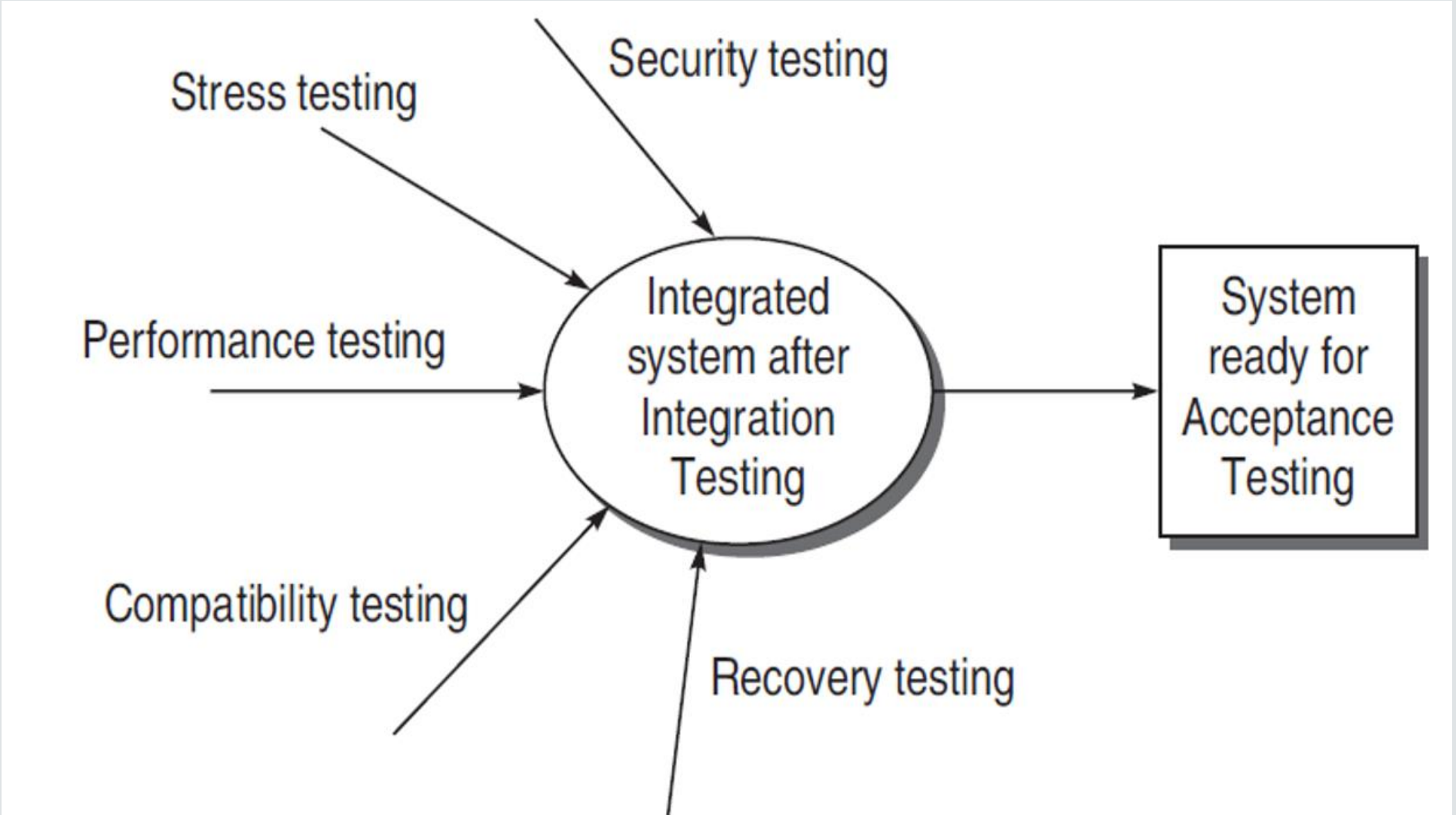




System Testing

- Test the whole system on various grounds
- The ground can be performance, security, maximum load etc
- It checks if program or system does not meet its original requirements an objectives as stated in the requirement specification.
- It focuses on testing the entire integrated system.

System Testing



Recovery Testing

- How good the developed software is when it faces a disaster
- Disaster can be unplugging the system, network ,stopping the database, software crash
- software systems (e.g. operating system, database management systems,etc.) must recover from programming errors, hardware failures, data errors or any disaster in the system

Testers Should Work On The Following Areas During Recovery Testing:

- **Restart** If there is a failure and we want to recover and start again
- Testers must ensure that all transactions have been reconstructed correctly and that all devices are in proper states.
- **Switchover** The ability of the system to switch to a new component must be tested.
- if there are standby components
- failure of one component, the standby takes over the control

Security Testing

- customers data has to be secured ensure that their functionality is properly implemented
- Internet users (personal data/information)is not secure the system loses its accountability.
- Security may include : controlling access to data
- encrypting data in communication,
- ensuring secrecy of stored data
- auditing security events
- effects of security breaches could be extensive and can cause loss of information, corruption information, misinformation, privacy violations, denial of service, etc.

Security Testing

Many different tasks are performed to manage software security risks, including

- Creating security abuse/misuse cases
- Listing security requirements
- Performing architectural risk analysis
- Building risk-based security test plans
- Performing security tests

Elements Of Security Testing

- **Confidentiality** protects against the disclosure of information to parties
- **Integrity** information at receiver side is not altered in transit or by anyone other
- **Authorization** process of determining that a requester is allowed to receive a service or perform an operation. Access control
- **Availability** Information must be kept available for authorized persons when they need it.
- **Non-repudiation** prevent the later denial that an action happened, or a communication took place, etc.
- authentication information combined with some timestamp.

Performance Testing

- How many concurrent users are expected for each target system server
- network appliance confi gurations time requirements
- testing team must use realistic databases
- Using realistic-size databases provides the following benefi ts:
- large dataset require signifi cant disk space and processor power
- Data transfer across a network, bandwidth may also be a consideration.

Performance Testing

Develop a high-level plan including requirements, resources, timelines, and milestones.

Develop a detailed performance test plan, including all dependencies and associated timelines.

- Choose test tools

Specify test data needed.

Confi gure the test environment

- Execute tests, probably repeatedly

Stress Testing

- It is a type of load testing
- system must be stress-tested
- system is put under loads beyond the limits so that the system breaks.
- Thus, stress testing tries to break the system under test by overwhelming its resources in order to fi nd the circumstances under which it will crash.
- In real-time systems, the entire threshold values and system limits must be noted carefully.
- real-time defense system, we must stress-test the system; otherwise, there may be loss of equipment as well as life.

Stress Testing: The Areas That May Be Stressed In A System Are

- Input transactions
- Disk space
- Output
- Communications
- Interaction with users

Compatibility/Conversion/Configuration Testing

- check the compatibility of a system being developed with different operating system, hardware and software configuration available

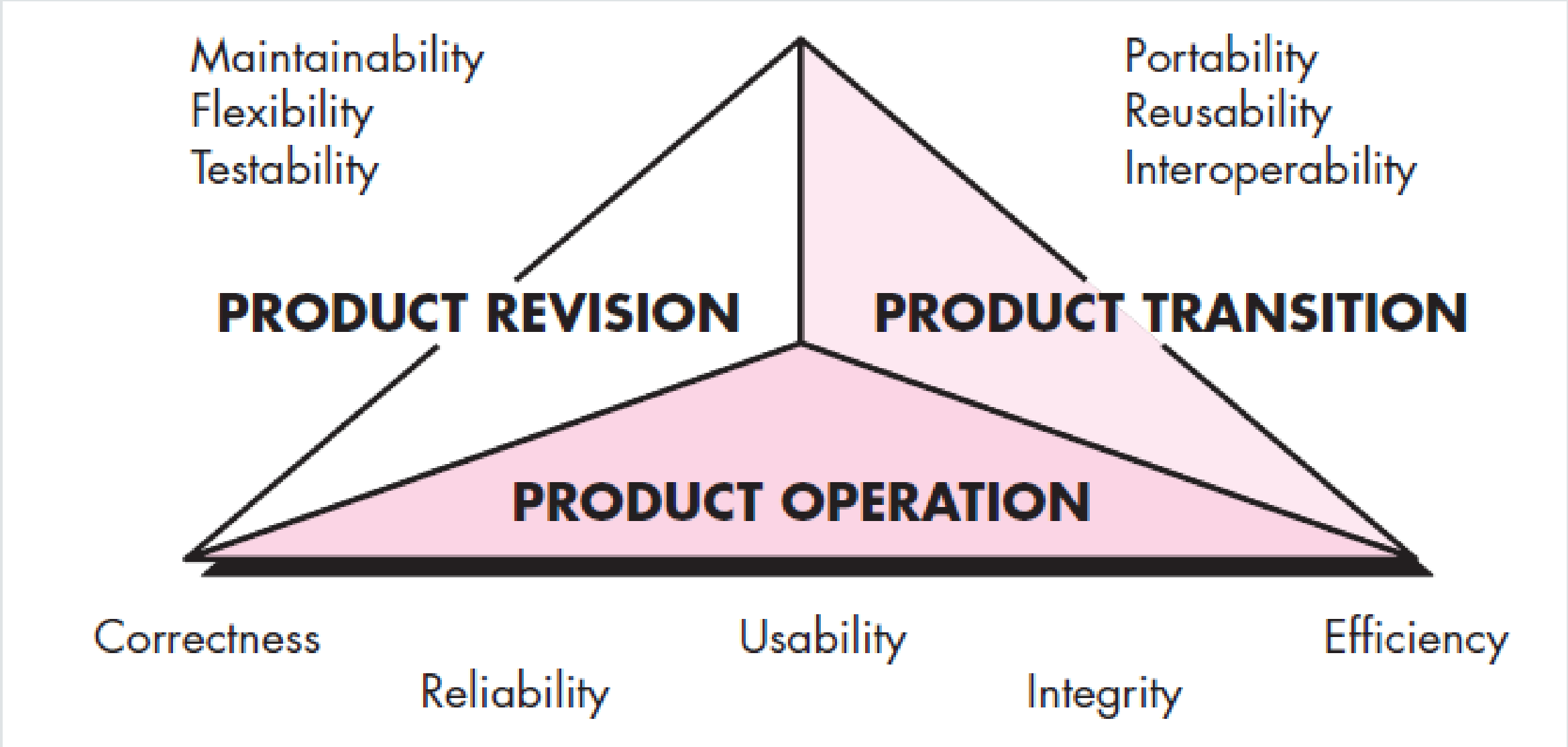
PRINCIPLES TO ACHIEVE SOFTWARE QUALITY

1. If a customer does not demand a quality product, he or she will probably not get one.
2. To consistently produce quality products, the developers must manage the quality of their work.
3. To manage product quality, the developers must measure quality.
4. The quality of a product is determined by the quality of the process used to develop it.
5. Since a test removes only a fraction of a product's defects, to get a quality product out of test you must put a quality product into test.
6. Quality products are only produced by motivated professionals who take pride in their work.

SOFTWARE QUALITY MODELS: MCCALL'S QUALITY FACTORS

- McCall software quality model was introduced in 1977.
- It incorporates many attributes
- These software quality factors, as shown in the above figure, here focus on three important aspects
- software product operational characteristics
- It includes requirements that directly affect the operation of the software
- These factors help in providing a better user experience.
- Product Revision Factors: It includes software quality factors, which are required for testing and maintenance of the software.
- Product Transition Factors: It includes three software quality factors, that allows the software to adapt to the change of environments

Software Quality Models: MCcall's Quality Factors

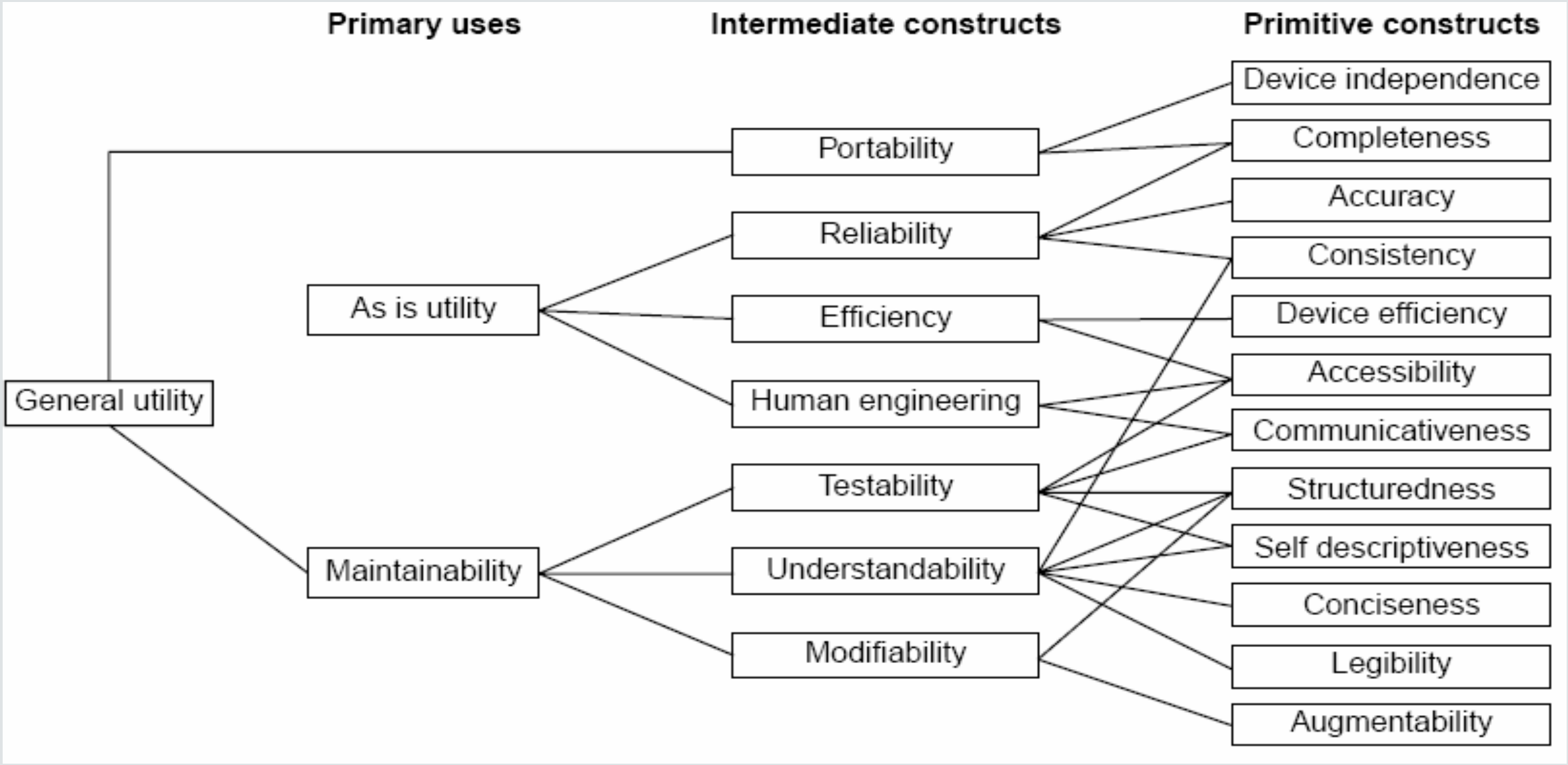


QUALITY FACTORS		
Product operation	Correctness	Ability To Fulfill Specification And Customer Requirements
	Reliability	Degree By Which Software Should Work As Per Requirement Precision
	Usability	Ability To Prepare The Valid Input And Interpret Correct Output
	Efficiency	Measure Of Computing Resources And Time Required By Program
	Integrity	Controlling Ability By Which Unauthorized Acces Can Be Prevented
Product Revision	Maintainability	Ability Required To Fix Bugs In Software
	Flexibility	Extent By Which It Is Allowed To Make Certain Modification In Program
	Testability	Ability To Check Function Is Working As Per Requirement
Product transition	Reusability	Ability For Software Component Can Be Reused By Other Program
	Portability	Ability For Software Work Even If Environment Gets Changed
	Interoperability	Ability Of System To Work With Other System

BOEHM'S SOFTWARE QUALITY MODEL

- In 1978, B.W. Boehm introduced his software quality model.
- The model represents a hierarchical quality model
- Boehm's quality model is based on a wider range of characteristics.
- The Boehm's model has three levels for quality attributes.
- These levels are divided based on their characteristics.
- primary uses (high level characteristics),
- intermediate constructs(mid-level characteristics)
- primitive constructs(primitive characteristics).
- As is utility –Extent to which, we can use software as-is.
  - Maintainability –Effort required to detect and fix an error during maintenance.
  - Portability –Effort required to change software to fit in a new environment.

Boehm Software Quality Model



Boehm's Software Quality Model

As-is Utility	Reliability	Self Containedness
		Accuracy
		Completeness
		Robustness/Integrity
		Consistency
	Efficiency	Accountability
		Device Efficiency
		Accessibility
	Human Engineering	Robustness/Integrity
		Accessibility
		Communicativeness
Portability		Device Independence
		Self Containedness
Maintainability	Testability	Accountability
		Communicativeness
		Self Descriptiveness
		Structuredness
	Understandability	Consistency
		Structuredness
		Conciseness
		Legibility
	Modifiability	Structuredness
		Augmentability
3 High-Level Characteristics	**7 Intermediate-Level Characteristics**	**15 Distinct Primitive Characteristics**

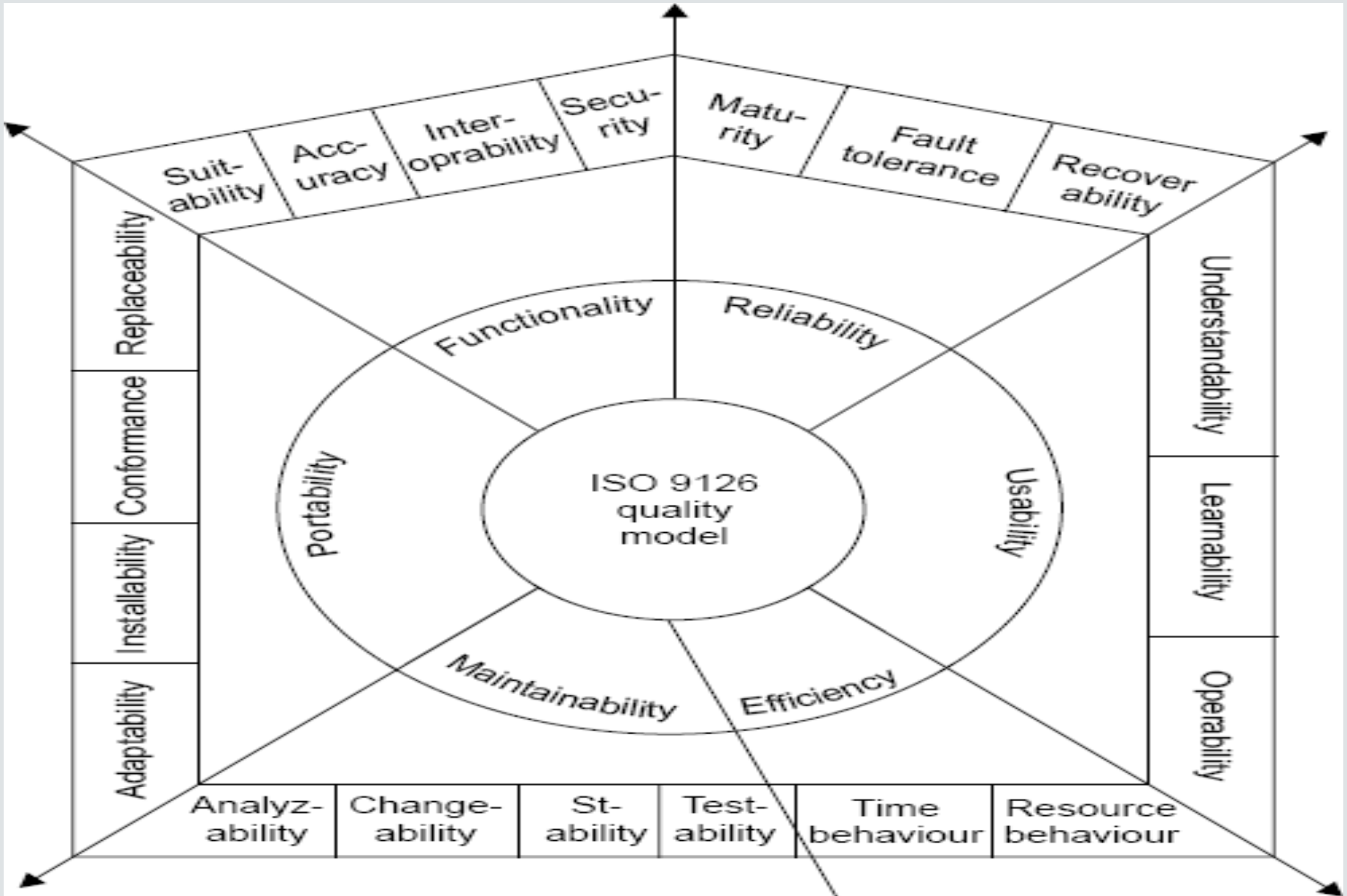
BOEHM'S SOFTWARE QUALITY MODEL

- The next level of Boehm's hierarchical model consists of seven quality factors associated with three primary uses, stated as below –
- Portability – Effort required to change software to fit in a new environment.
- Reliability – Extent to which software performs according to requirements.
- Efficiency – Amount of hardware resources and code required to execute a function.
- Usability – Extent of effort required to learn, operate and understand functions of the software.
- Testability – Effort required to verify that software performs its intended functions.
- Understandability – Effort required for a user to recognize logical concept and its applicability.
- Modifiability – Effort required to modify a software during maintenance phase.

Boehm's Software Quality Model

- Boehm further classified characteristics into Primitive constructs
- Boehm's model is an improvised version of McCall's model
- Testability is broken down into:- accessibility, communicativeness, structuredness and self descriptiveness.
- Advantages :
  - It satisfy the needs of the user.
  - It focuses on software maintenance cost effectiveness.

ISO 9126 Quality Model

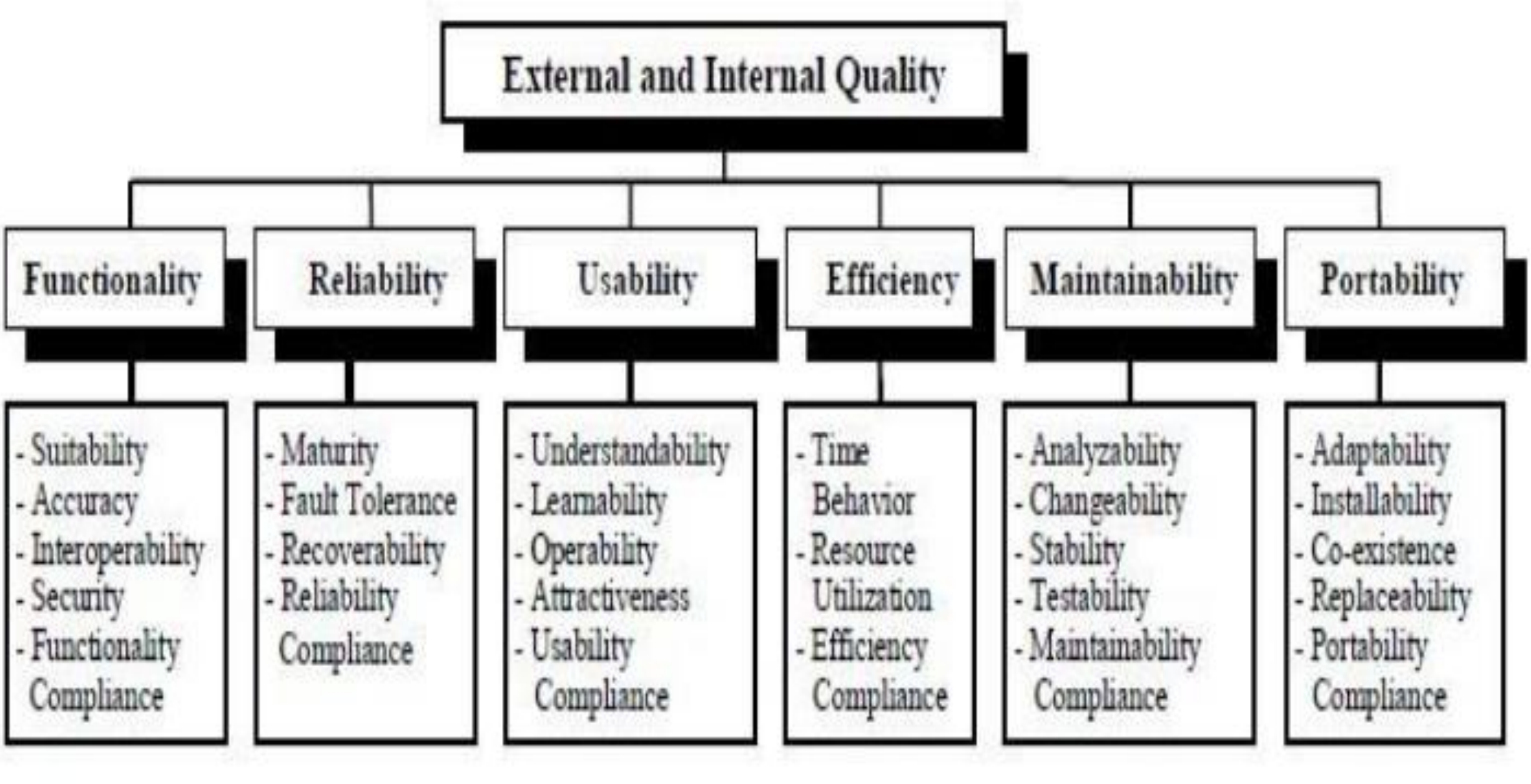


ISO 9126 QUALITY FACTORS

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

ISO 9126 HAS DEVELOPED FOLLOWING ATTRIBUTES FOR SOFTWARE QUALITY		
FACTOR NAME	**QUALITY FACTORS**	**DESCRIPTION**
Functionality	Sustainability, Accuracy, Compliance, Security	Ability By Which Software Satisfies Needs
Reliability	Maturity, Fault Tolerance, Recoverability	Degree By Which Software Will Be Available
Usability	Understandability, Learnability, Operability	Indicated Usefullness Of Software
Efficiency	Time Behaviour, Resource Behaviour	Measure Of Computing Resource And Time Required By Program To Perform
Maintainability	Anayzability,stability,testability, Changability	Ability To Fix Bug In Software
Portability	Adaptability ,Instanability, Replacability	Ability Of Software To Work Properly

ISO 9126 Quality Factors



ISO 9126 Quality Factors

Characteristic/ Attribute	Short Description of the Characteristics and the concerns Addressed by Attributes
Functionality	Characteristics relating to achievement of the basic purpose for which the software is being engineered
• Suitability	The presence and appropriateness of a set of functions for specified tasks
• Accuracy	The provision of right or agreed results or effects
• Interoperability	Software's ability to interact with specified systems
• Security	Ability to prevent unauthorized access, whether accidental or deliberate, to program and data.
Reliability	Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time
• Maturity	Attributes of software that bear on the frequency of failure by faults in the software

ISO 9126 Quality Factors

• Fault tolerance	Ability to maintain a specified level of performance in cases of software faults or unexpected inputs
• Recoverability	Capability and effort needed to reestablish level of performance and recover affected data after possible failure.
Usability	Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated implied set of users.
• Understandability	The effort required for a user to recognize the logical concept and its applicability.
• Learnability	The effort required for a user to learn its application, operation, input and output.
• Operability	The ease of operation and control by users.
Efficiency	Characteristic related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

ISO 9126 Quality Factors

• Time behavior	The speed of response and processing times and throughout rates in performing its function.
• Resource behavior	The amount of resources used and the duration of such use in performing its function.
Maintainability	Characteristics related to the effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functions specifications.
• Analyzability	The effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
• Changeability	The effort needed for modification, fault removal or for environmental change.
• Stability	The risk of unexpected effect of modifications.
• Testability	The effort needed for validating the modified software.

ISO 9126 Quality Factors

Portability	Characteristics related to the ability to transfer the software from one organization or hardware or software environment to another.
• Adaptability	The opportunity for its adaptation to different specified environments.
• Installability	The effort needed to install the software in a specified environment.
• Conformance	The extent to which it adheres to standards or conventions relating to portability.
• Replaceability	The opportunity and effort of using it in the place of other software in a particular environment.

CAPABILITY MATURITY MODELS (CMM)

- The U.S. Software Engineering Institute (SEI) initiated a study of ways to assess the capabilities o software contractors.
- The outcome of this capability assessment was the SEI Software Capability Maturity Model (CMM).
- The Software CMM was followed by a range of other capability maturity models, including the Peopl Capability Maturity Model (P-CMM) and the Systems Engineering Capability Model.
- Other organizations have also developed comparable process maturity models.
- The SPICE (Software Process Improvement and Capability dEtermination) approach to capabilit assessment and process improvement is more flexible than the SEI model.

CAPABILITY MATURITY MODEL (CMM)

- measure the quality of a process by recognizing it on a maturity scale.
- maturity of a process directly affects the progress of a project and its results
- SEI developed the CMM

CMM Structure

- CMM consists of fi ve maturity levels.
- Maturity levels consist of key process areas (KPAs) which are organized by common features.
- KPA identifi es a cluster of related activities
- features in turn consist of key practices
- Key practices describe the activities to be done

CMM STRUCTURE

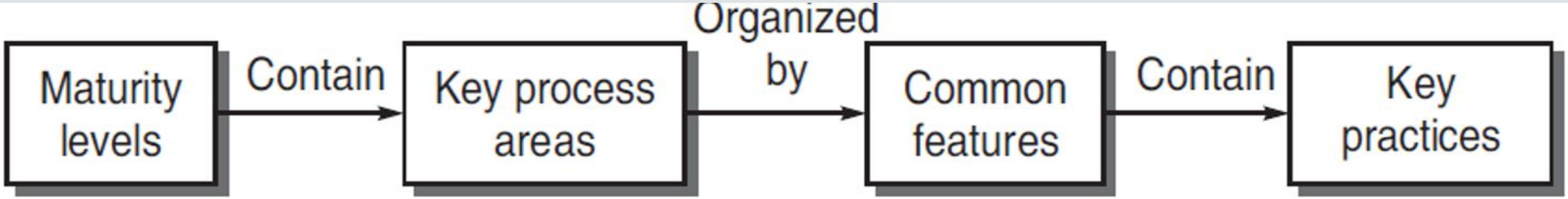


Figure 13.7(a) CMM Structure

Maturity levels	Indicate	Process capability
Key process areas	Achieve	Goals
Common features	Address	Implementation/Institutionalization
Key practices	Describe	Infrastructure/Activities

CMM Structure

- structure of CMM is hierarchical in nature
- such that if one organization wishes to be on maturity level 3, then it must fi rst achieve maturity level 2

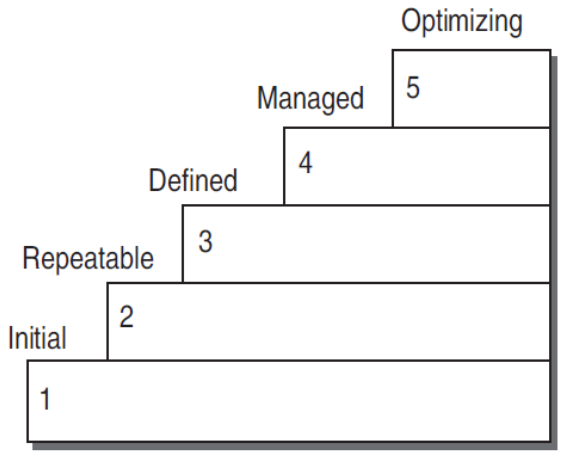


Figure 13.8 Maturity levels

Table 13.1 Process capability at each maturity level

Maturity level	Process Capability
1	—
2	Disciplined process
3	Standard consistent process
4	Predictable process
5	Continuously improving process

CMM MATURITY LEVELS

- A brief description of the fi ve maturity levels

Initial : At this level, there is no control on development progress

- success is purely dependent on individual efforts.
- There is no stable environment for development.
- The projects exceed the budget and schedule.

Repeatable Earlier project successes are used here

- Basic project management processes are established
- cost, schedule, and other parameters can be tracked. organization-wide processes still do not exist.

Defi ned: Project management as well as process management starts.

- The standard processes are defi ned with documentations.

CMM MATURITY LEVELS

- All level 2 criteria have been achieved.
- Process area is defined as per organizational policy

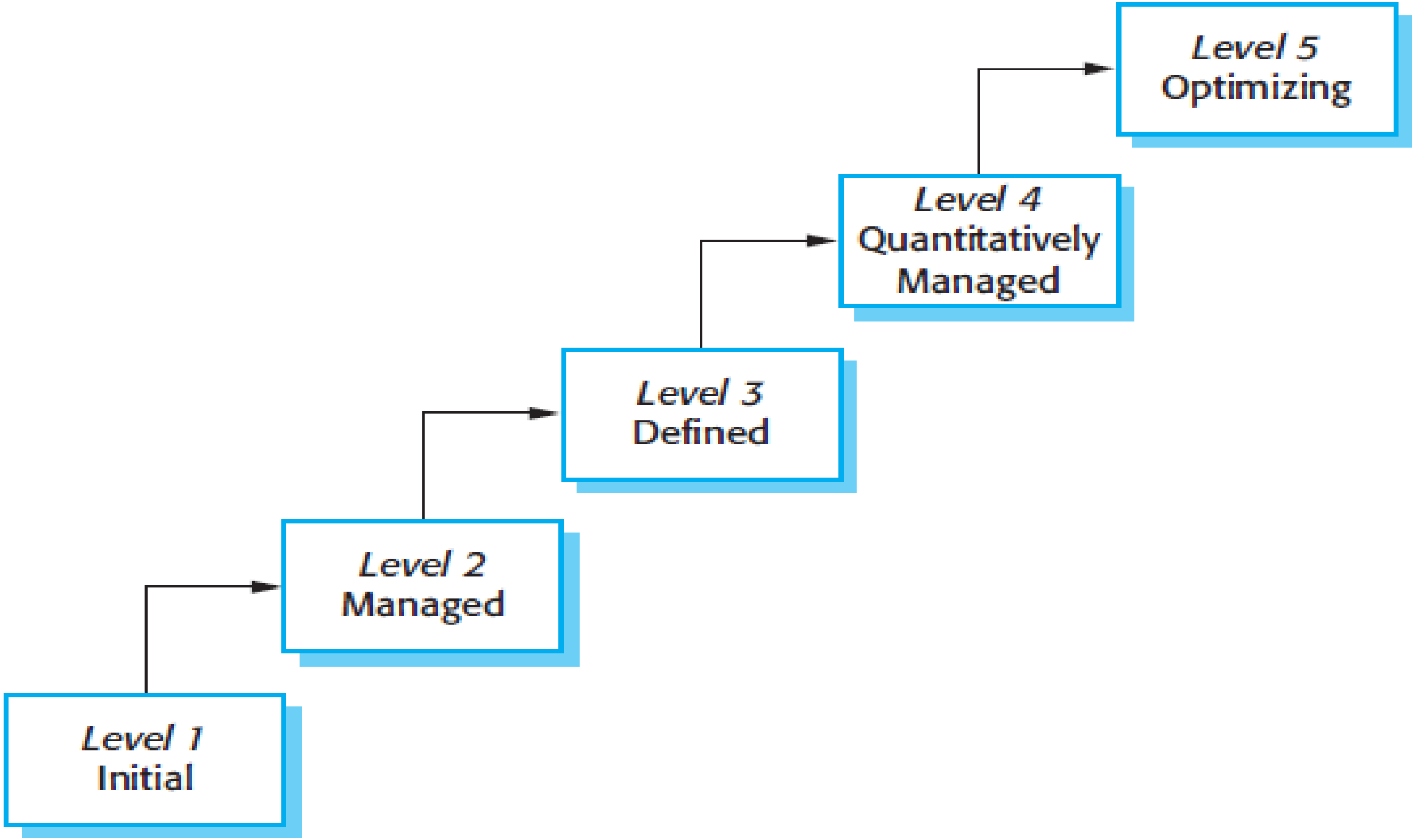
Managed: processes can be monitored and controlled

- quality goals are also set for the processes.
- processes are predictable and controlled.
- All level 3 criteria have been satisfied.
- Process area is controled and improved
- The main objective of quality is established

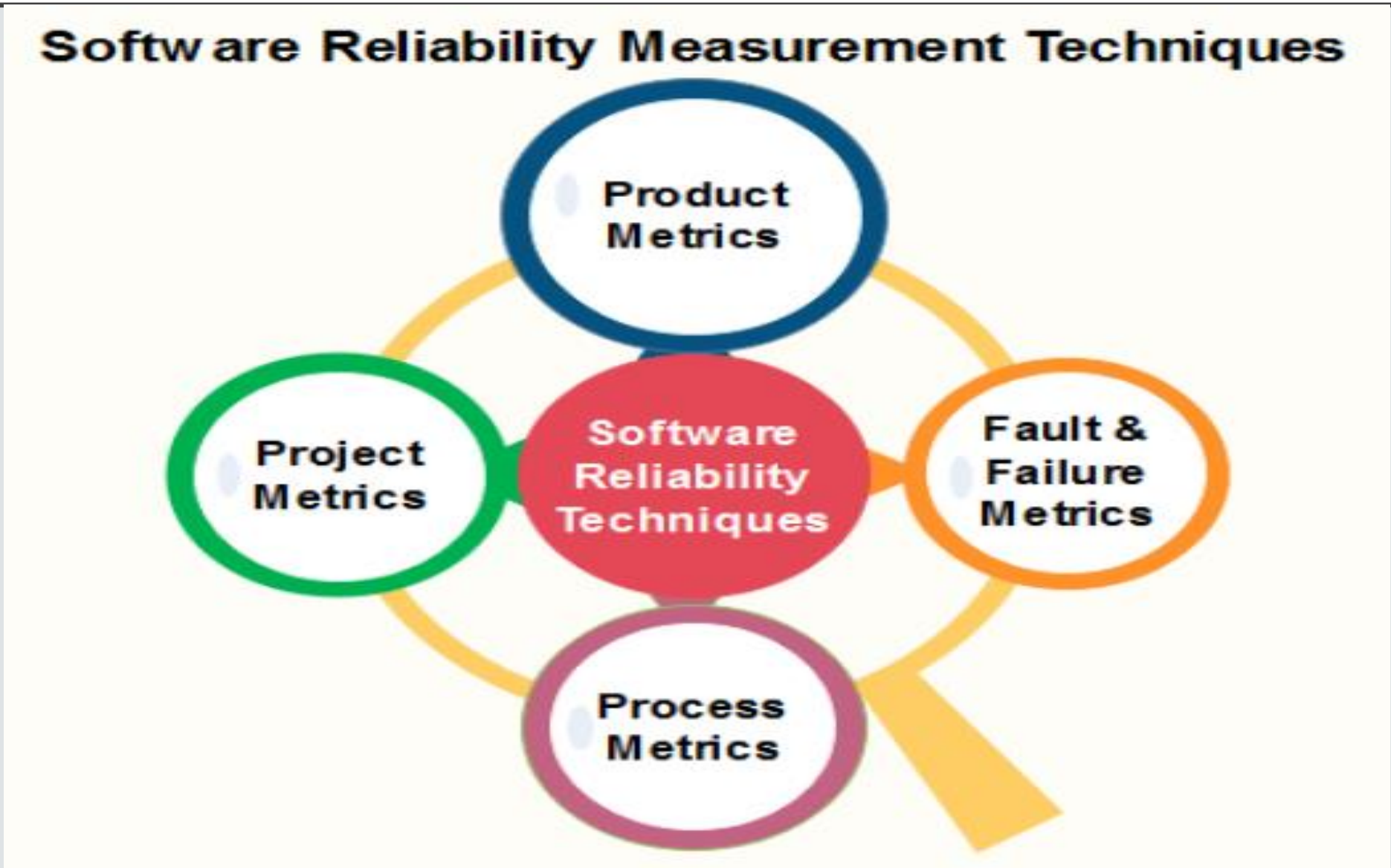
Optimizing : The process is continually improved through incremental and innovative technological changes or improvements.

- process improvement objectives are identifi ed, evaluated, and deployed.

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution
Quantitatively managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management
Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Integrated Supplier Management Risk Management Decision Analysis and Resolution Organizational Environment for Integration Integrated Teaming
Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
Performed		


Level 1
Initial
Level 2
Managed
Level 3
Defined
Level 4
Quantitatively
Managed
Level 5
Optimizing

SOFTWARE METRICS



PRODUCT METRICS

1.Software size. Lines of Code (LOC), or LOC in thousands (KLOC),

2.Function point metric is a technique to measure the functionality of proposed software development based on the count of inputs, outputs, master files, inquires, and interfaces.

3.Test coverage metric It is portion of software that is successfully verified or tested.

4.Complexity It determine the complexity of a program's control structure by simplifying the code into a graphical representation.

5.Quality metrics measure the quality at various steps of software product development. An vital quality metric is Defect Removal Efficiency (DRE)..

PROCESS METRICS

- So process metrics can be used to estimate, monitor, and improve the reliability and quality of software.
- Process metrics describe the effectiveness and quality of the processes that produce the software product.

Examples are:

- The effort required in the process
- Time to produce the product
- Effectiveness of defect removal during development
- Number of defects found during testing
- Maturity of the process

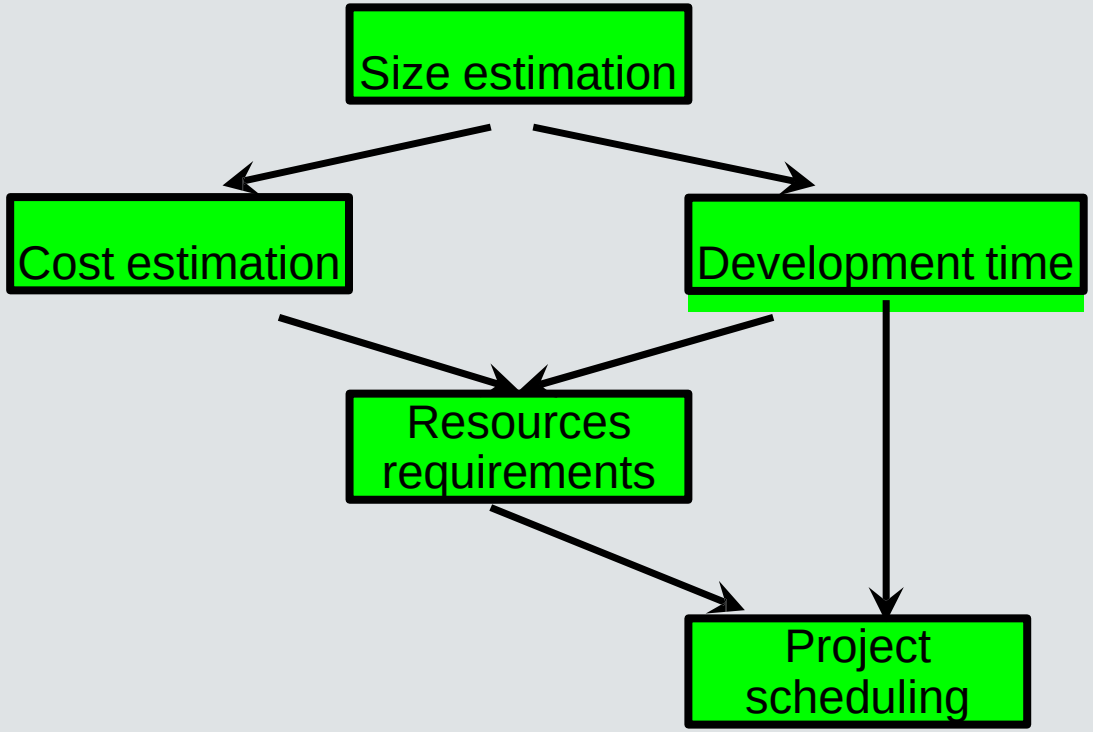
SOFTWARE PROJECT PLANNING

In order to conduct a successful software project, we must understand:

- Scope of work to be done
- The risk to be incurred
- The resources required
- The task to be accomplished
- The cost to be expended
- The schedule to be followed

SOFTWARE PROJECT PLANNING

Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.



Size estimation: lines Of Code (LOC)

Lines of Code (LOC)

- If LOC is simply a count of the number of lines then figure shown below contains 18 LOC .

- When comments and blank lines are ignored, the program in figure 2 shown below contains 17 LOC.

```
1.  int. sort (int x[ ], int n)
2.  {
3.  int i, j, save, im1;
4.  /*This function sorts array x in ascending order */
5.  If (n<2) return 1;
6.  for (i=2; i<=n; i++)
7.  {
8.  im1=i-1;
9.  for (j=1; j<=im; j++)
10. if (x[i] < x[j])
11. {
12. Save = x[i];
13. x[i] = x[j];
14. x[j] = save;
15. }
16. }
17. return 0;
18. }
```

Size estimation: lines Of Code (LOC)

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declaration, and executable and non-executable statements".

This is the predominant definition for lines of code used by researchers. By this definition, figure shown above has 17 LOC.

Function Point Analysis (FPA)

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

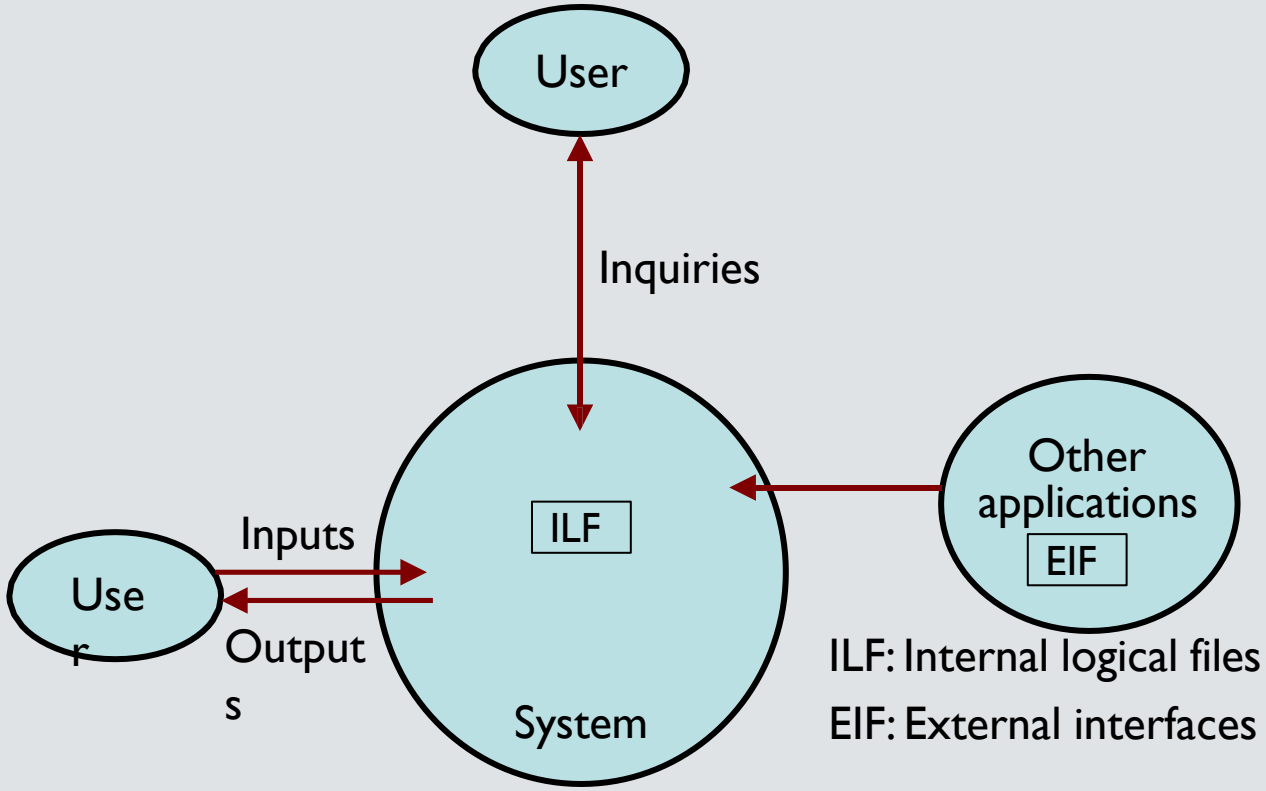
Function Point Analysis (FPA)

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the
▪ Outputs	:	system information leaving the
▪ Enquiries	:	system
▪ Internal logical files	:	requests for instant access to information
▪ External interface files	:	information held within the system
		information held by other system that is used by the system being analyzed.

Function Point Analysis (FPA)

The FPA functional units are shown in figure given below:



Functional Units With Weighting Factors

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Function Point Analysis (FPA)

Table 2: UFP calculation table

Functional Units	Count Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)		Low x 3 =		
		Average x 4 =		
		High x 6 =		
External Outputs (EOs)		Low x 4 =		
		Average x 5 =		
		High x 7 =		
External Inquiries (EQs)		Low x 3 =		
		Average x 4 =		
		High x 6 =		
External logical Files (ILFs)		Low x 7 =		
		Average x 10 =		
		High x 15 =		
External Interface Files (EIFs)		Low x 5 =		
		Average x 7 =		
		High x 10 =		
Total Unadjusted Function Point Count				

Weighting Factors & Unadjusted Function Point (UFP)

The weighting factors are identified for all functional units and multiplied with the functional units accordingly.

The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

Unadjusted Function Point (UFP)

The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

Where i indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

Function Point Analysis (FPA)

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions noted in table 3.

Function Point Analysis (FPA)

Table 3 : Computing function points.

Rate each factor on a scale of 0 to 5.

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require on line data entry ?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated on line ?
9. Is the inputs, outputs, files, or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by the user ?

Function Points Compute The Following Important Metrics

Functions points may compute the following important metrics:

Productivity	=	FP / persons-months
Quality	=	Defects / FP
Cost	=	Rupees / FP
Documentation	=	Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Functions Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point method is also being developed.

Function Points Example

Consider a project with the following functional units:

Number of user inputs	= 50
Number of user outputs	= 40
Number of user enquiries	= 35
Number of user files	= 06
Number of external interfaces	= 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Function Points Example

Solution

We know

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = $(0.65 + 0.01 \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

Function Points Example

An application has the following:

10 low external inputs, 12 high external outputs, 20 low

internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Function Points Example

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

Function Points Example

Consider a project with the following parameters.

(i) External Inputs:
- (a)10 with low complexity
- (b)15 with average complexity
- (c)17 with high complexity

(ii) External Outputs:
- (a)6 with low complexity
- (b)13 with high complexity

(iii) External Inquiries:
- (a) 3 with low complexity
- (b) 4 with average complexity
- (c) 2 high complexity

Function Points Example

(iv) Internal logical files:

(a)2 with average complexity

(b)1 with high complexity

(v) External Interface files:

(a) 9 with low complexity

In addition to above, system requires

i. Significant data communication

ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Function Points Example

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity	Complexity	Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	
	15	Average x 4	=	60	
	17	High x 6	=	102	192
External Outputs (EOs)	6	Low x 4	=	24	
	0	Average x 5	=	0	
	13	High x 7	=	91	115
External Inquiries (EQs)	3	Low x 3	=	9	
	4	Average x 4	=	16	
	2	High x 6	=	12	37
External logical Files (ILFs)	0	Low x 7	=	0	
	2	Average x 10	=	20	
	1	High x 15	=	15	35
External Interface Files (EIFs)	9	Low x 5	=	45	
	0	Average x 7	=	0	
	0	High x 10	=	0	45
Total Unadjusted Function Point Count					424

Function Points Example

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

CAF = $(0.65 + 0.01 \times \Sigma F_i)$

= $(0.65 + 0.01 \times 41)$

= 1.06

FP = UFP x CAF

= 424 x 1.06

= 449.44

Hence **FP = 449**

Token Count

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta_1 + \eta_2$$

where

η : vocabulary of a program

η_1 : number of unique operators

η_2 : number of unique operands

Token Count

The length of the program in the terms of the total number of tokens used is

$N = N_1 + N_2$

where

N : program length

N_1 : total occurrences of operators

N_2 : total occurrences of operands

Token Count

Volume

$$V = N * \log_2 \eta$$

The unit of measurement of volume is the common unit for size "bits". It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

Program Level

$$L = V^* / V$$

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

Token Count

Program Difficulty

$$D = 1 / L$$

As the volume of an implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

Effort

$$E = V / L = D * V$$

The unit of measurement of E is elementary mental discriminations.

Token Count

- Estimated Program Length

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

$$\hat{N} = 14 \log_2 14 + 10 \log_2 10$$

= 53.34 + 33.22 = 86.56

The following alternate expressions have been published to estimate program length.

$$N_J = Log_2(\eta_1!) + \log_2(\eta_2!)$$

Token Count

$$N_B = \eta_1 Log_2 \eta_2 + \eta_2 \log_2 \eta_1$$

$$N_c = \eta_1 \sqrt{\eta_1} + \eta_2 \sqrt{\eta_2}$$

$$N_s = (\eta \log_2 \eta) / 2$$

The definitions of unique operators, unique operands, total operators and total operands are not specifically delineated.

Token Count

Consider the sorting program in Fig. 2 of chapter 4. List out the operators and operands and also calculate the values of software science measures like η, N, V, E, λ *etc.*

Solution

The list of operators and operands is given in table 2.

Operators	*Occurrences*	*Operands*	*Occurrences*
int	4	SORT	1
( )	5	x	7
,	4	n	3
[ ]	7	i	8
if	2	j	7
<	2	save	3

(Contd.)...

Token Count

;	11	im1	3
for	2	2	2
=	6	1	3
–	1	0	1
< =	2	—	—
+ +	2	—	—
return	2	—	—
{ }	3	—	—
$\eta_1 = 14$	$N_1 = 53$	$\eta_2 = 10$	$N_2 = 38$

Operators and operands of sorting program

Token Count

Here N_1=53 and N_2=38. The program length $N=N_1+N_2=91$

Vocabulary of the program $\eta = \eta_1 + \eta_2 = 14 + 10 = 24$

Volume $V = N \times \log_2 \eta$

= 91 x $\log_2 24$ = 417 bits

The estimated program length $\hat{N}$ of the program

= 14 $\log_2 14$ + 10 $\log_2 10$

= 14 * 3.81 + 10 * 3.32

= 53.34 + 33.2 = 86.45

Token Count

Conceptually unique input and output parameters are represented by η_2^*

$\eta_2^* = 3$ {x: array holding the integer to be sorted. This is used both as input and output}.

{N: the size of the array to be sorted}.

The potential volume $V^* = 5 \log_2 5 = 11.6$

Since $L = V^* / V$

Token Count

$$= \frac{11.6}{417} = 0.027$$

D = I / L

$$= \frac{1}{0.027} = 37.03$$

Estimated program level

$$\hat{L} = \frac{2}{\eta_1} \times \frac{\eta_2}{N_2} = \frac{2}{14} \times \frac{10}{38} = 0.038$$

Token Count

We may use another formula

$$\hat{V} = V \times \hat{L} = 417 \times 0.038 = 15.67$$

$$\hat{E} = V / \hat{L} = \hat{D} \times V$$

=417 / 0.038 = 10973.68

Therefore, 10974 elementary mental discrimination are required to construct the program.

$$T = E / \beta = \frac{10974}{18} = 610\,\text{seconds} = 10\,\text{minutes}$$

This is probably a reasonable time to produce the program, which is very simple

What Is Software Reliability?

"Software reliability means operational reliability.

Who cares how many bugs are in the program?

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

What Is Software Reliability?

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error.

"It is the probability of a failure free operation of a program for a specified time in a specified environment".

What Is Software Reliability?

Failures and Faults

A fault is the defect in the program that,

when fault executed under particular conditions, causes a failure.

Four General Ways Of Characterising Failure Occurrences In Time:

1. time of failure
2. time interval between failures,
3. cumulative failure experienced up to a given time,
4. failures experienced in a time interval.

Time Based Failure Specification

Failure Number	Failure Time (sec)	Failure interval (sec)
1	8	8
2	18	10
3	25	7
4	36	11
5	45	9
6	57	12
7	71	14
8	86	15
9	104	18
10	124	20
11	143	19
12	169	26
13	197	28
14	222	25
15	250	28

Failure Based Failure Specification

Time (sec)	Cumulative Failures	Failure in interval (30 sec)
30	3	3
60	6	3
90	8	2
120	9	1
150	11	2
180	12	1
210	13	1
240	14	1

Probability Distribution At Times T_A T_B

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
0	0.10	0.01
1	0.18	0.02
2	0.22	0.03
3	0.16	0.04
4	0.11	0.05
5	0.08	0.07
6	0.05	0.09
7	0.04	0.12
8	0.03	0.16
9	0.02	0.13

Probability Distribution At Times T_a And T_b

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
10	0.01	0.10
11	0	0.07
12	0	0.05
13	0	0.03
14	0	0.02
15	0	0.01
Mean failures	3.04	7.77

LOGARITHMIC POISSON EXECUTION TIME MODEL

- It is one of the **Model** is a **software reliability growth model (SRGM)**
- It predict software failure rates over time.
- It helps estimate software reliability based on observed failure data during testing.
- **Overview of the Logarithmic Poisson Model**
- The model assumes that **software failures occur according to a Poisson process**.
- It considers that **the failure intensity decreases logarithmically as the number of detected failures increases**.
- It is useful for analyzing software reliability **during testing and operational phases**.

LOGARITHMIC POISSON EXECUTION TIME MODEL

2. Mathematical Representation

The expected cumulative number of failures at time t, denoted as $m(t)$, is given by:

$$m(t) = a \log(1 + bt)$$

Where:

- a = Total number of expected failures in the software
- b = Model parameter representing the rate of failure detection
- t = Time or test execution time

The failure intensity function $\lambda(t)$ is:

$$\lambda(t) = \frac{ab}{(1 + bt)}$$

ASSUMPTIONS OF LOGARITHMIC POISSON EXECUTION TIME MODEL

- Failures occur independently and follow a **Poisson distribution**.
- The failure rate **decreases** as more bugs are fixed.
- The number of failures detected follows a **logarithmic growth** pattern.

APPLICATIONS OF LOGARITHMIC POISSON EXECUTION TIME MODEL

- **Software Testing**: Predicting software reliability during the testing phase.
- **Reliability Prediction**: Estimating remaining failures before release.
- **Resource Allocation**: Planning testing efforts efficiently

JMETER

- JMeter is an open-source
- It is Java-based software
- It is primarily used for performance testing, load testing, and functional testing of web applications
- It allows testers to simulate heavy user loads on a system to measure its performance under stress by creating a large number of concurrent users
- JMeter is freely available and developed by the Apache Software Foundation.
- It provides a user-friendly interface
- JMeter can run on any platform with a compatible Java Development Kit (JDK).

APPLICATIONS OF APACHE JMETER

- They are not limited as it covers the wide range of systems mentioned below:

1. **Web Application Testing:** Web Application covers many frameworks web testing can be divided into **Load Testing, Stress Testing, Functional Testing, API Testing,** etc.

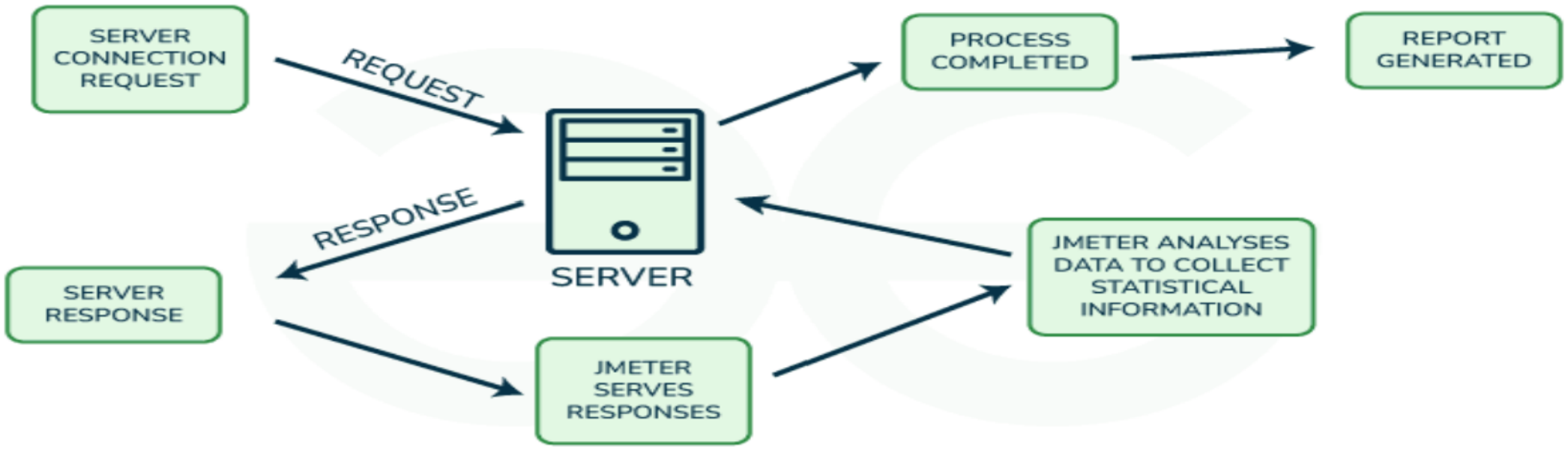
2. **Mobile Application Testing:** Wide range features of JMeter supports various protocols, scripting which effectively helps in Mobile Testing like **Performance Testing, Battery Consumption Testing,** etc.

3. **Database Testing:** It is the process in we evaluate how a database responds under different workloads and conditions.

4. **Other Application:** Apart from Web , Mobile and Database JMeter can be used for other applications like Desktop Applications

APACHE JMETER WORKING

- JMeter is a Java based application
- It simulates a group of users and sends request to a target server.
- It returns statistics which shows the functionality and performance of the server.



APACHE JMETER FEATURES

Features of JMeter that make it so powerful are mentioned below:

- **Performance Testing:** It plays a vital role in testing any application. JMeter tool has this feature of Performance Testing.
- **Load Testing:** It assess how the application will perform when it faces a load of users and activity
- **Functional Testing:** It checks if the functionality of the application is up to the mark.
- It ensures that the application behaves as expected and delivers the desired features and functionalities to the user.

APACHE JMETER FEATURES

- **Scripting:** Scripting allows us to create custom test scenarios
- JMeter is able to test wide range scenarios and corner cases.
- **Distributed Testing:** It is technique which allows to distribute the load for testing into multiple machines rather than a single machine.
- These multiple machines are controlled by a single parent machine. It is poweful technique used JMeter check the real time consequences of load.
- **Graphical Reporting:** Graphical Reporting is where we use Interactive visual to represent the report

ADVANTAGES OF APACHE JMETER

- *It is an Open source Application that is free and even easy to use.*
- *It is flexible which means can be used according to the requirement.*
- *It is a powerful application and can be used to test a wide range of systems*

BUGZILLA

- Bugzilla is a widely used, open-source
- It is web-based tool primarily designed for bug tracking
- It is originally developed and maintained by the Mozilla Foundation
- It is essentially acting as a centralized platform for tracking defects and ensuring software quality.

BUGZILLA

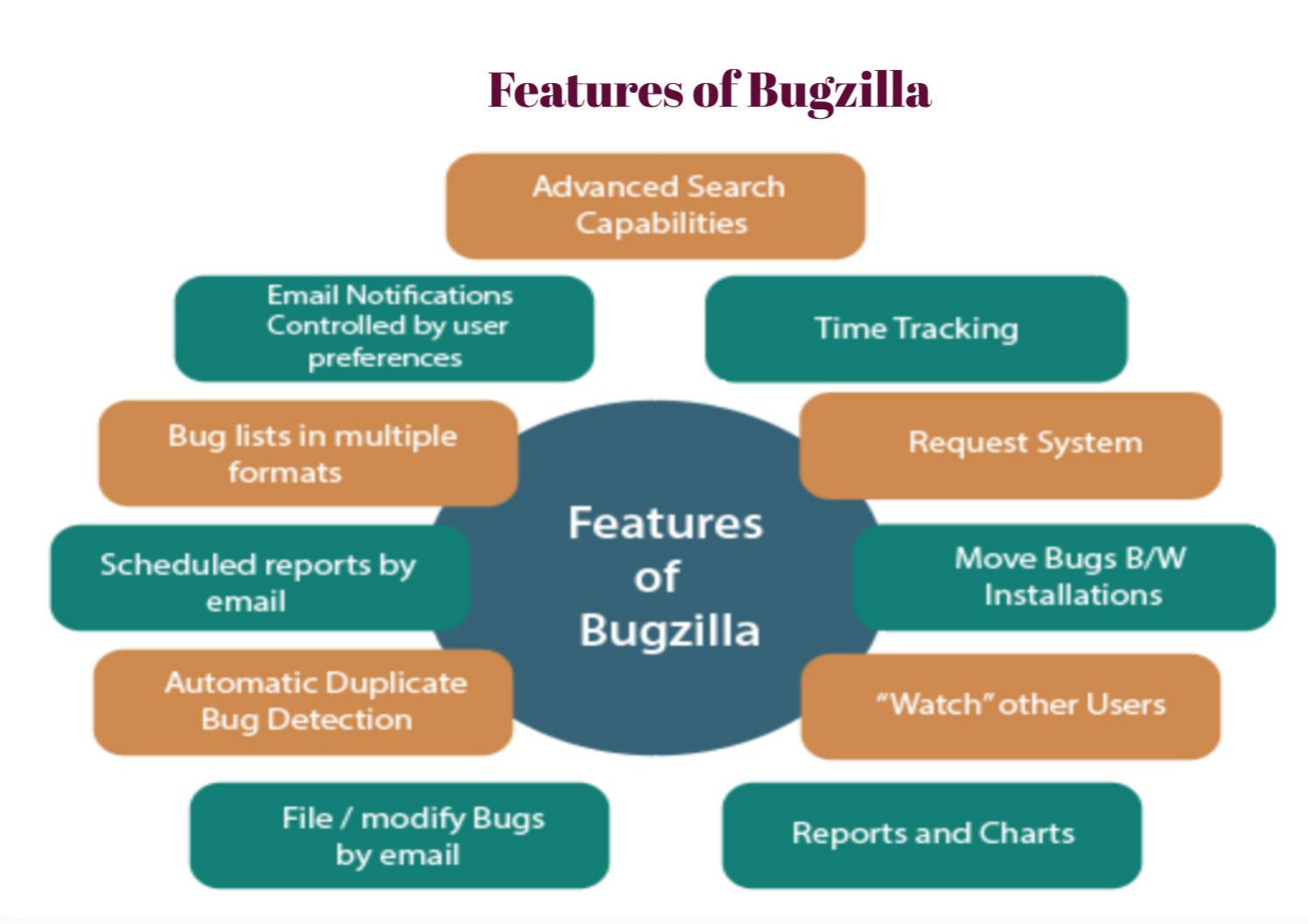
Bugzilla is a bug tracking tool
It helps to track the issues related to their product.

Bugzilla tool is written in Perl language
It uses MySQL database.

It can also be used as a test management tool
It can be linked with other Test case management tools

It is an open source tool, i.e., this tool is available to the users at a free of cost.

BUGZILLA


Features of Bugzilla
Advanced Search Capabilities
Email Notifications Controlled by user preferences
Time Tracking
Bug lists in multiple formats
Request System
Features of Bugzilla
Scheduled reports by email
Move Bugs B/W Installations
Automatic Duplicate Bug Detection
"Watch" other Users
File / modify Bugs by email
Reports and Charts

SELENIUM TESTING TOOL

WHAT IS TEST AUTOMATION?

- The process of converting manual test cases into test scripts by using automation tools is called test automation.
- The test scripts can be executed on the targeted application whenever required.

Advantages

- Saves time by executing the script without any manual effort.
- We can reduce the cost to the company.
- Test scripts are repeatable.
- Bugs can be identified and accuracy maintained.
- Test report will be generated by the tool and we can ensure quality.

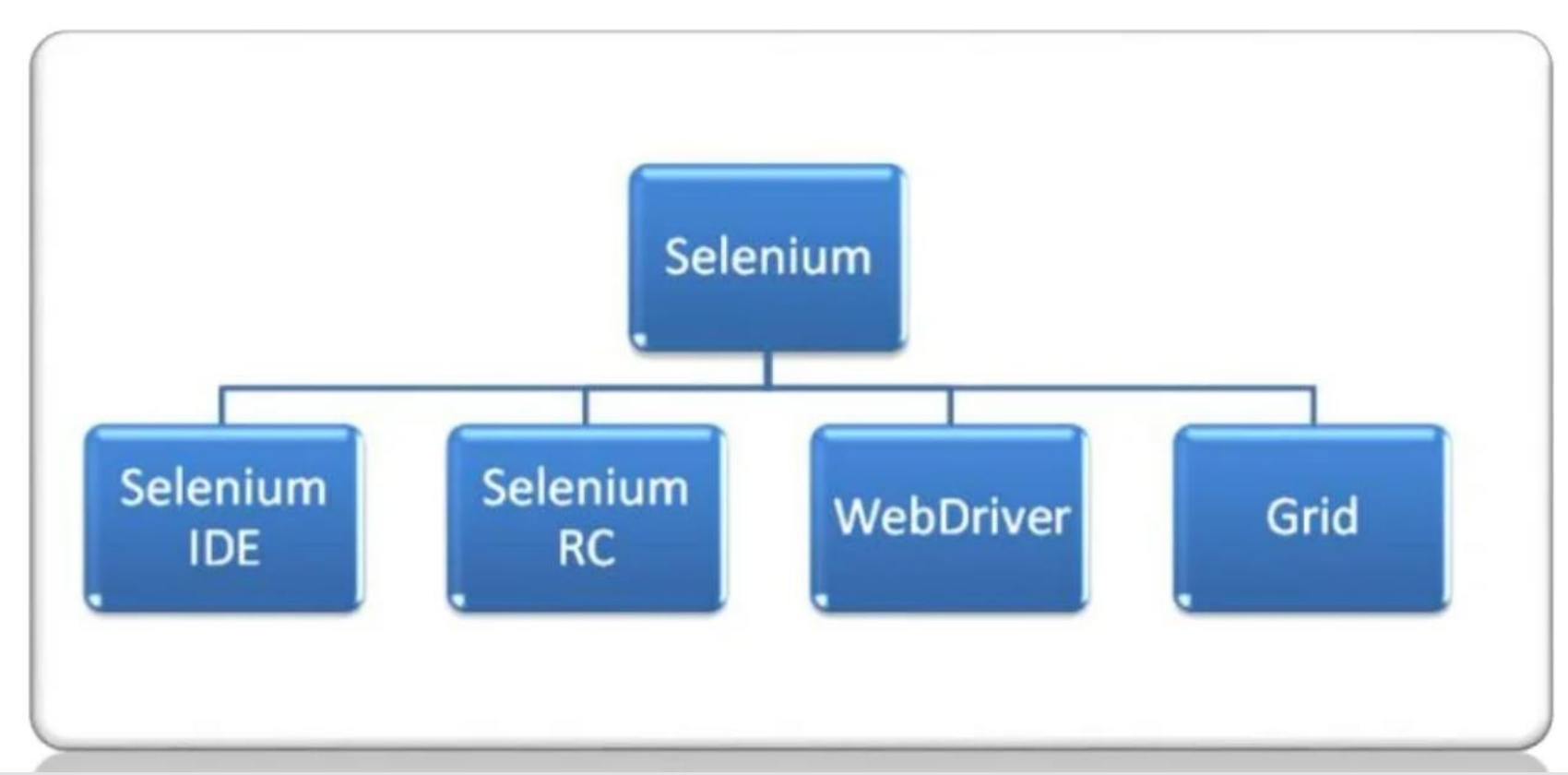
WHAT IS SELENIUM?

- Selenium is a set of open source testing tools that supports rapid development of test automation for web-based applications.
- It is used for functional regression testing.
- It works on multiple browsers and operating systems.
- Helpful to distribute/scale your test cases across multiple platforms.
- It uses JavaScript.
- Supports languages like HTML, Java, PHP, Perl, Python, Ruby and C#.
- It can be integrated with several defect tracking tools.
- It is easy to use.

HISTORY

- Was developed by Jason Huggins in 2004. He was a working on a in house project in Thoughtworks. He started working on Selenium Core.
- Dan Fabulich and Nelson Sproul, created Selenium RC.
- Shinya Kasatani in Japan became interested in Selenium, and realised that he could wrap the core code into an IDE module into the Firefox browser, and be able to record tests as well as play them back in the same plugin.

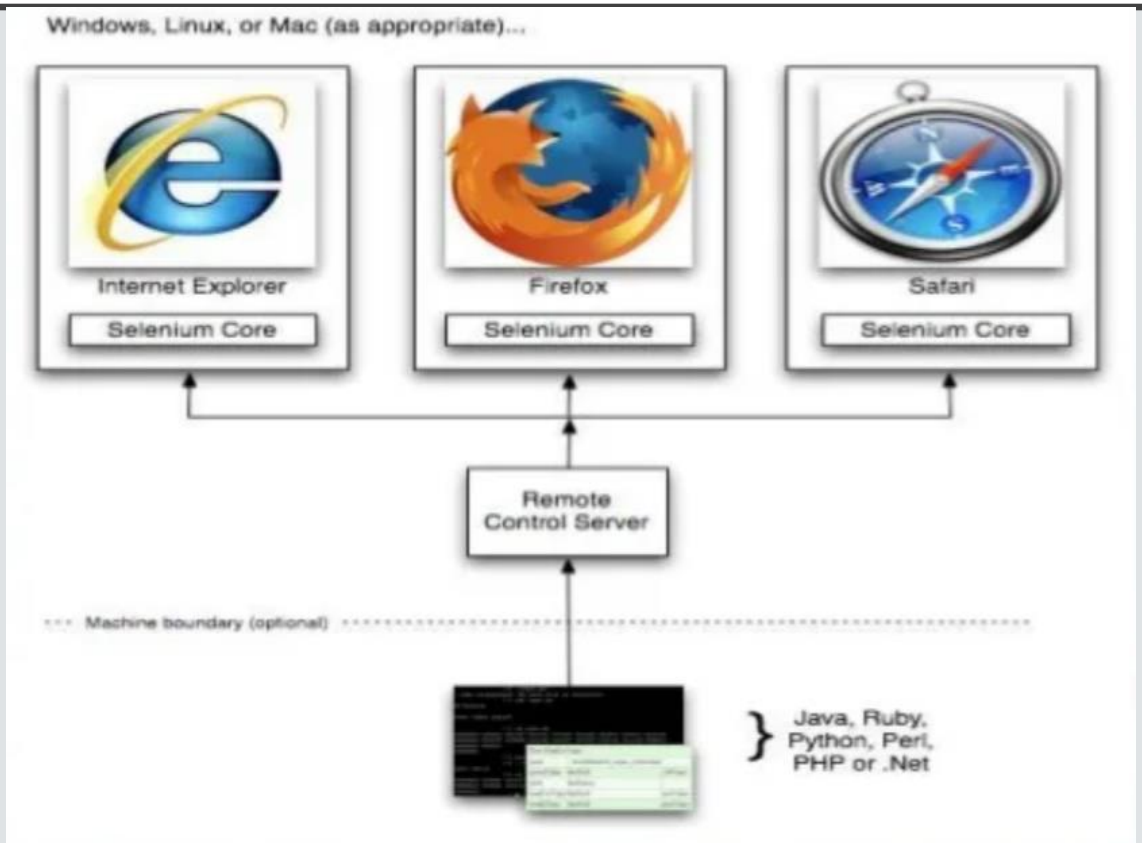
SELENIUM COMPONENTS



SELENIUM REMOTE CONTROL(RC)

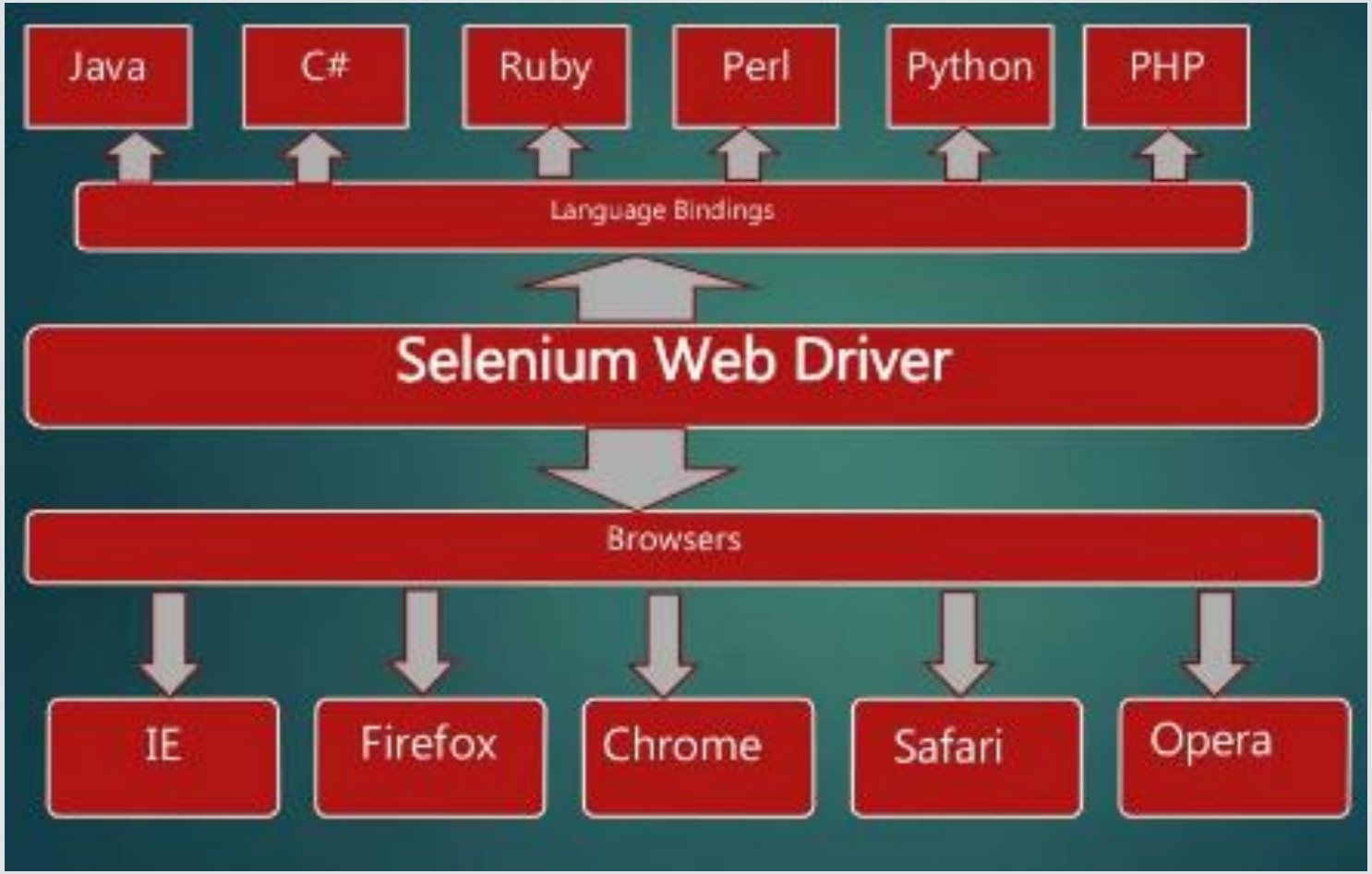
- Selenium-RC provides an API (Application Programming Interface) and library for each of its supported languages HTML Java, C#, Perl, PHP, Python, and Ruby,
- This ability to use Selenium-RC with a high-level programming language to develop test cases also allows the automated testing to be integrated with a project's automated build environment.
- Also known as Selenium 1.

HOW DOES SELENIUM RC WORK?



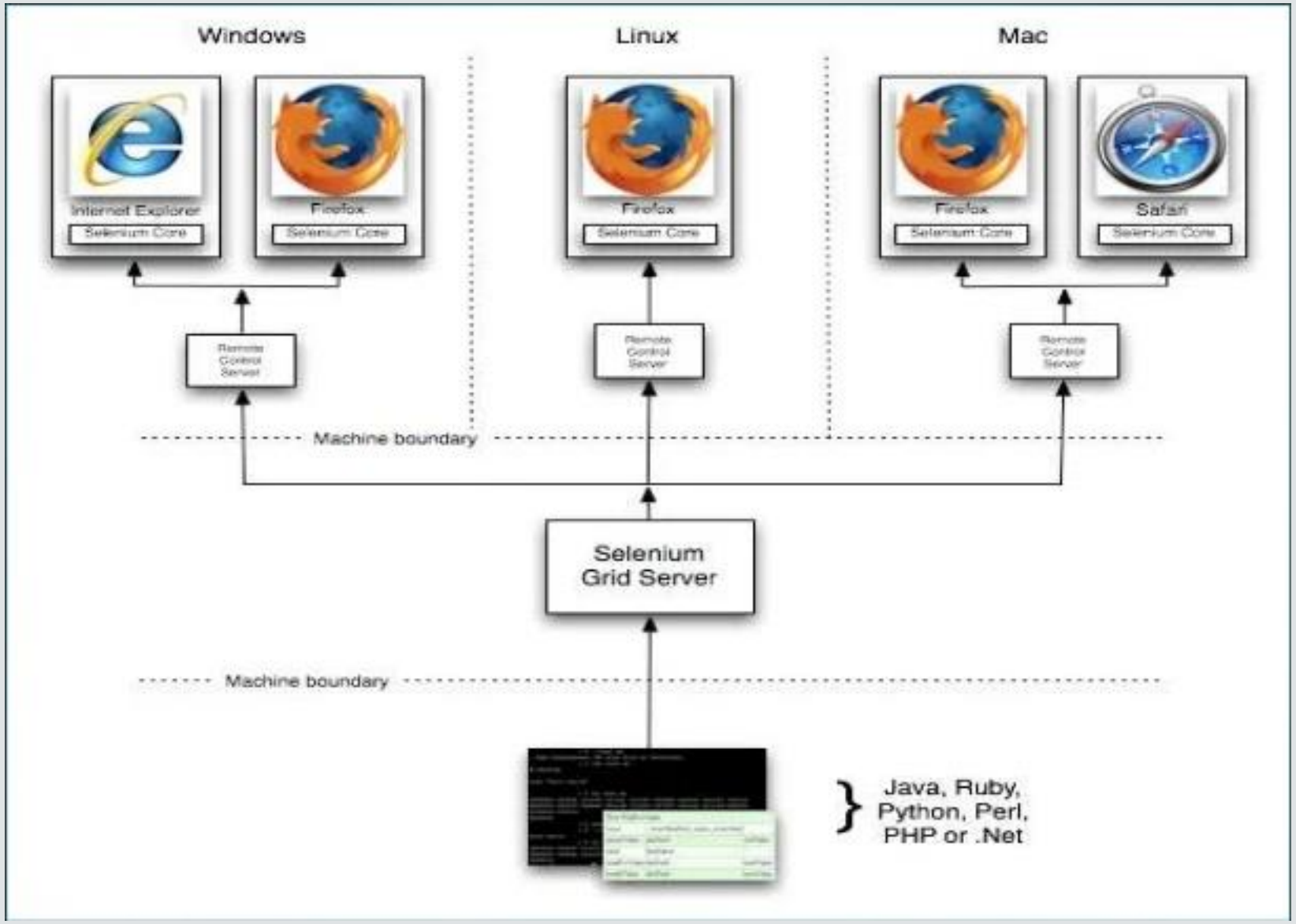
SELENIUM WEBDRIVER

- Selenium WebDriver is the successor of Selenium Remote Control which has been officially deprecated.
- Selenium WebDriver is a collection of language speafic bindings to drive a browser.
- WebDriver is designed in a simpler and more concise programming interface along with addressing some limitations in the Selenium-RC APL.
- Also known as Selenium 2
  - Selenium 1+ Selenium Web Driver = Selenium 2


Java
C#
Ruby
Perl
Python
PHP
Language Bindings
Selenium Web Driver
Browsers
IE
Firefox
Chrome
Safari
Opera

SELENIUM GRID

- Selenium Grid can run large test suites and test that must be run in multiple environments.
- Tests can be run in parallel with simultaneous execution (different tests on different remote machines).
- It allows for running your tests in a distributed test execution environment.
- Used to run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
- It reduces the time it takes for the test suite to complete a test pass.

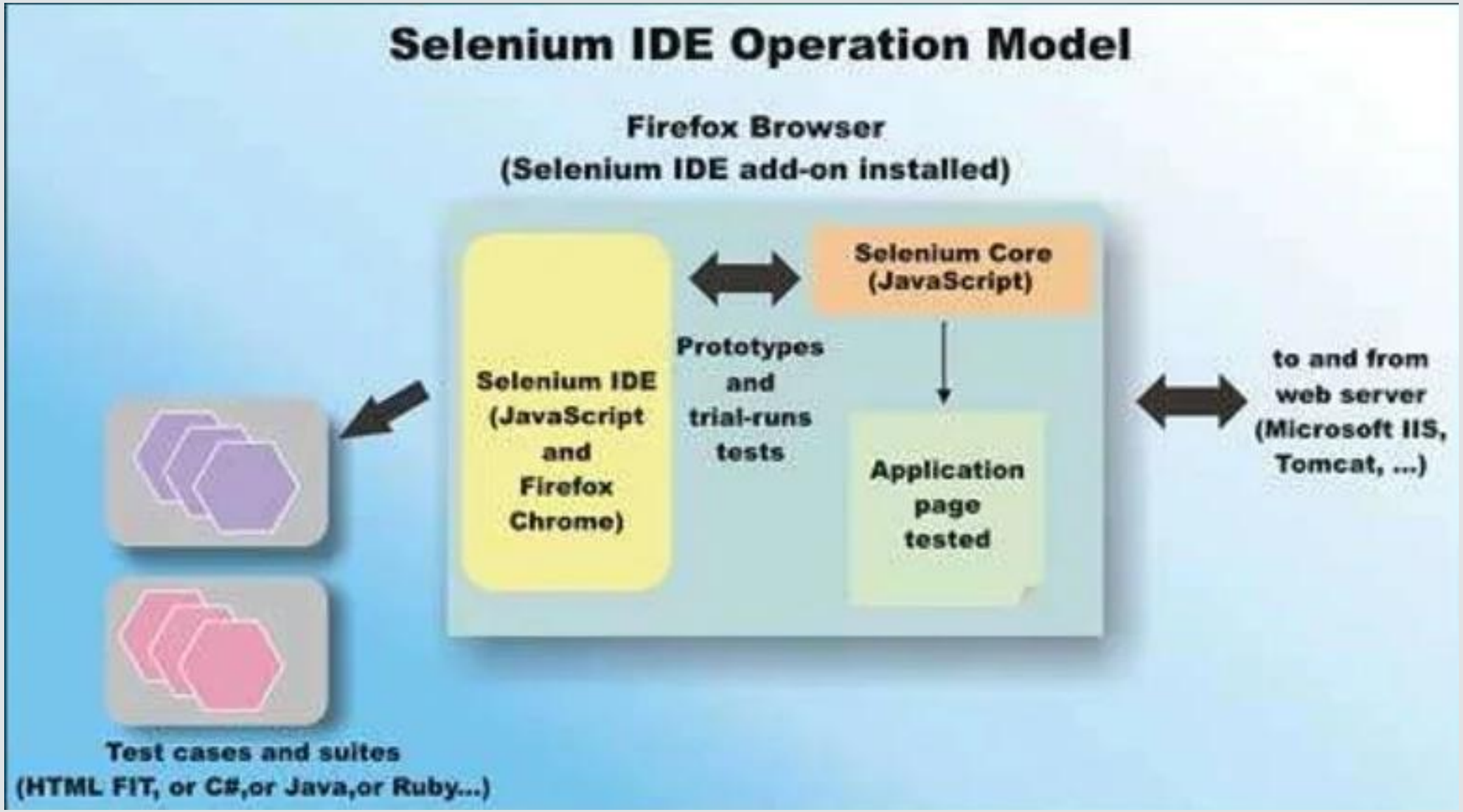

Windows
Linux
Mac
Internet Explorer
Selenium Core
Firefox
Selenium Core
Firefox
Selenium Core
Firefox
Selenium Core
Safari
Selenium Core
Machine boundary
Selenium
Grid Server
Machine boundary
Java, Ruby,
Python, Perl,
PHP or .Net

SELENIUM IDE

- If you want to create quick bug reproduction scripts or create scripts to aid in automation-aided exploratory testing, then Selenium IDE is the best.
- Selenium IDE is an integrated development environment for Selenium scripts.
- It's an add on with Firefox.
- It has record and playback functionality. Though it can record only record on Firefox, it can be modified to run in webdriver and RC.
- The tests can be modified, assertions can be added and test suite can be created.

FEATURES OF SELENIUM IDE

- Easy record and playback.
- Intelligent field selection will use IDs, names, or XPath as needed.
- Autocomplete for all common Selenium commands.
- Walk through tests.
- Debug and set breakpoints.
- Save tests as HTML, Ruby scripts, or any other format .
- Support for Selenium user-extensions.js file.
- Option to automatically assert the title of every page.


Selenium IDE Operation Model
Firefox Browser
(Selenium IDE add-on installed)
Selenium IDE
(JavaScript
and
Firefox
Chrome)
Prototypes
and
trial-runs
tests
Selenium Core
(JavaScript)
Application
page
tested
to and from
web server
(Microsoft IIS,
Tomcat, ...)
Test cases and suites
(HTML FIT, or C#,or Java,or Ruby...)

WHY SELENIUM OVER OTHER TOOLS?

- Most powerful Open Source Automation tool available.
- Flexible with support to many languages
- Highly extensible
- Platform Support-Provides support on wide range of OS compared to any other tool
- Parallel Testing-Supports parallel Testing
- Usability - Easy to use. ALM Integration - Provides integration with several bug tracking tools.

SELENESE

- Selenium commands, often called selenese. The set of these commands are nothing but test script.
- If you want to write test scripts for any application, initially you need to integrate Selenium with Java by using Eclipse. (check the Integration doc).
- After completion of the integration, First we need to create the selenium object as below.

```
DefaultSelenium selenium=new DefaultSelenium("System address", "Port number of selenium", "Browser to launch", "url").
```

Generally we use the below selenium commands to work on any application:

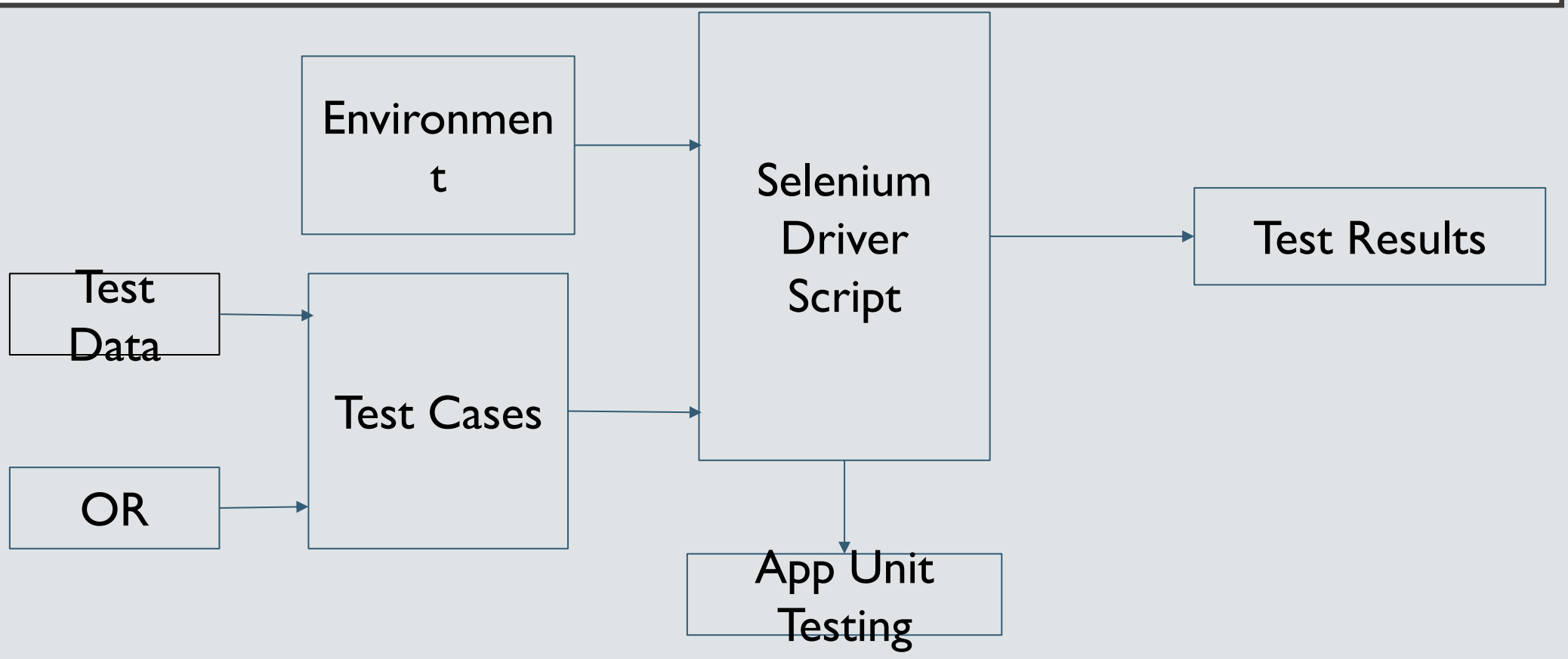
- Start(): To launch the Browser.
- Open(): To open the url.
- Close(): To kill or close the Browser.
- windowMaximize(): To maximize the window.
- Type(): To enter some text into a text box.
- Click(): To click on Button, Radio button and Link.
- Select(): To select a value or label from combo box or list box or drop down.
- Check(): To check the check box.

- selectPopUp(): To identify the pop up window.
- selectWindow(): To identify the child window.
- selectFrame(): To identify the frame.
- getAlert(): To Click ok on alert box.
- getConfirmation(): To click ok on confirmation message.
- chooseCancelOnNextConfirmation(): To click Cancel on next displayed confirmation message.
- chooseOkOnNextConfirmation(): To click Cancel on next displayed confirmation message.

TESTING WITH SELENIUM

- We can perform Regression testing with selenium by writing reusable scripts in the form of functions or methods.
- We can perform Retesting with looping concepts.
- We can validate the application by using conditional statements.
- We can perform database testing.
- Reporting of the test results.
- Error handling, particularly unexpected errors.
- Test case grouping.
- Re-execution of failed test cases.
- Screenshot capture of test failures.

SELENIUM FRAMEWORK



SELENIUM FRAMEWORK FLOW:

- The execution of the script starts, based on the 'Selenium Driver Script.
- 'Selenium Driver Script' takes the Url from Environment folder and Test cases form 'Test Cases' folder as an input.
- In the 'Test Cases' folder, all the test cases will be stored and these test cases will take the test data from 'Test Data' folder and/or from 'Object Repository' folder.
- After completion of the execution, the results will be stored in 'Results' folder. It contains
  - Result as 'Pass' or 'Fail' with the entire test cases what we have given for execution.
  - All the screenshots of executed test cases.
- If test cases are not available in the 'Test Cases' folder then the framework will still export the results into result folder saying that there are no test cases are available.