ASTANA IT UNIVERSITY

Software Quality Assurance and Testing

Lecture 1

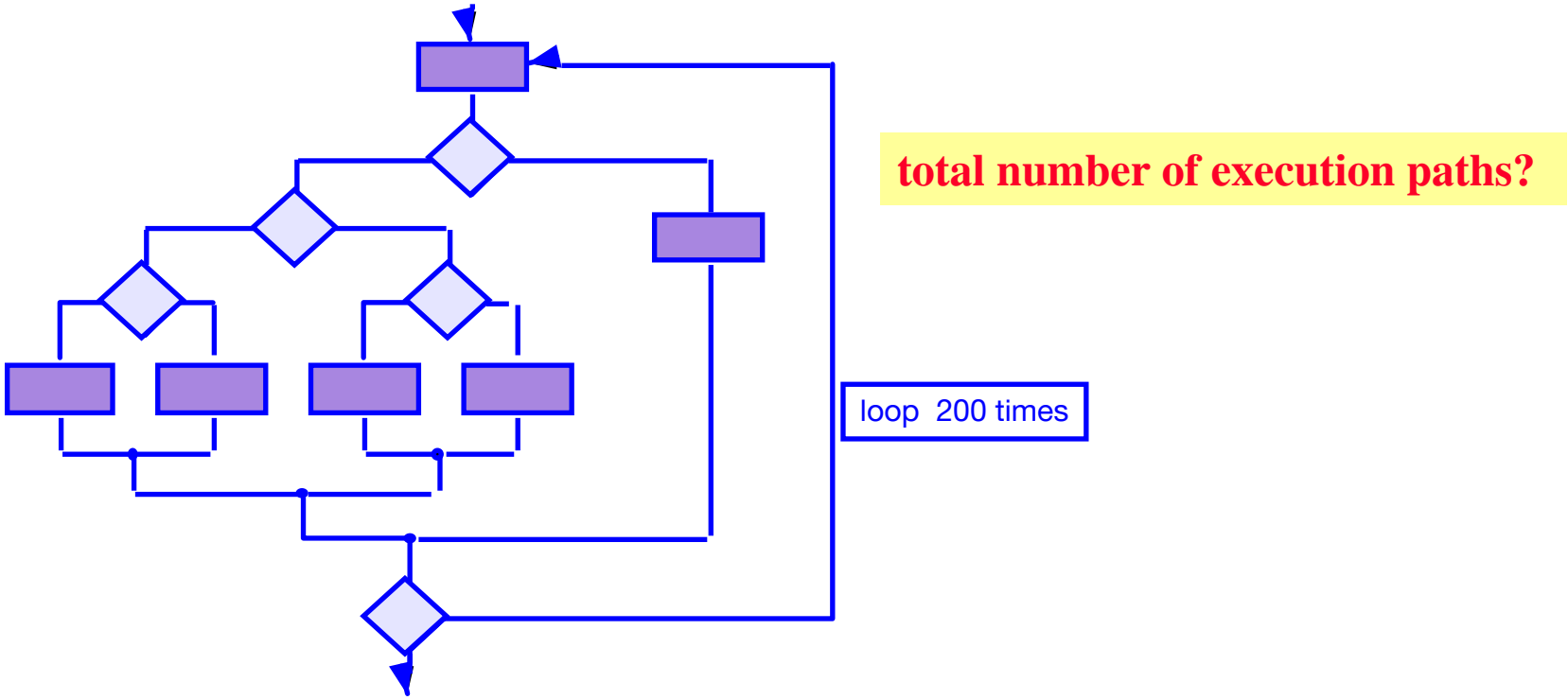
Outline

- Software Quality Assurance and Testing Fundamentals
- Software Testing Types: Functional and Non-Functional Testing Types
- Black-Box testing
- White-Box Testing
- Sanity Testing
- Smoke Testing
- Regression Testing
- Bugs. Defects.
- Bug Life Cycle
- Bug Reporting and Tracking

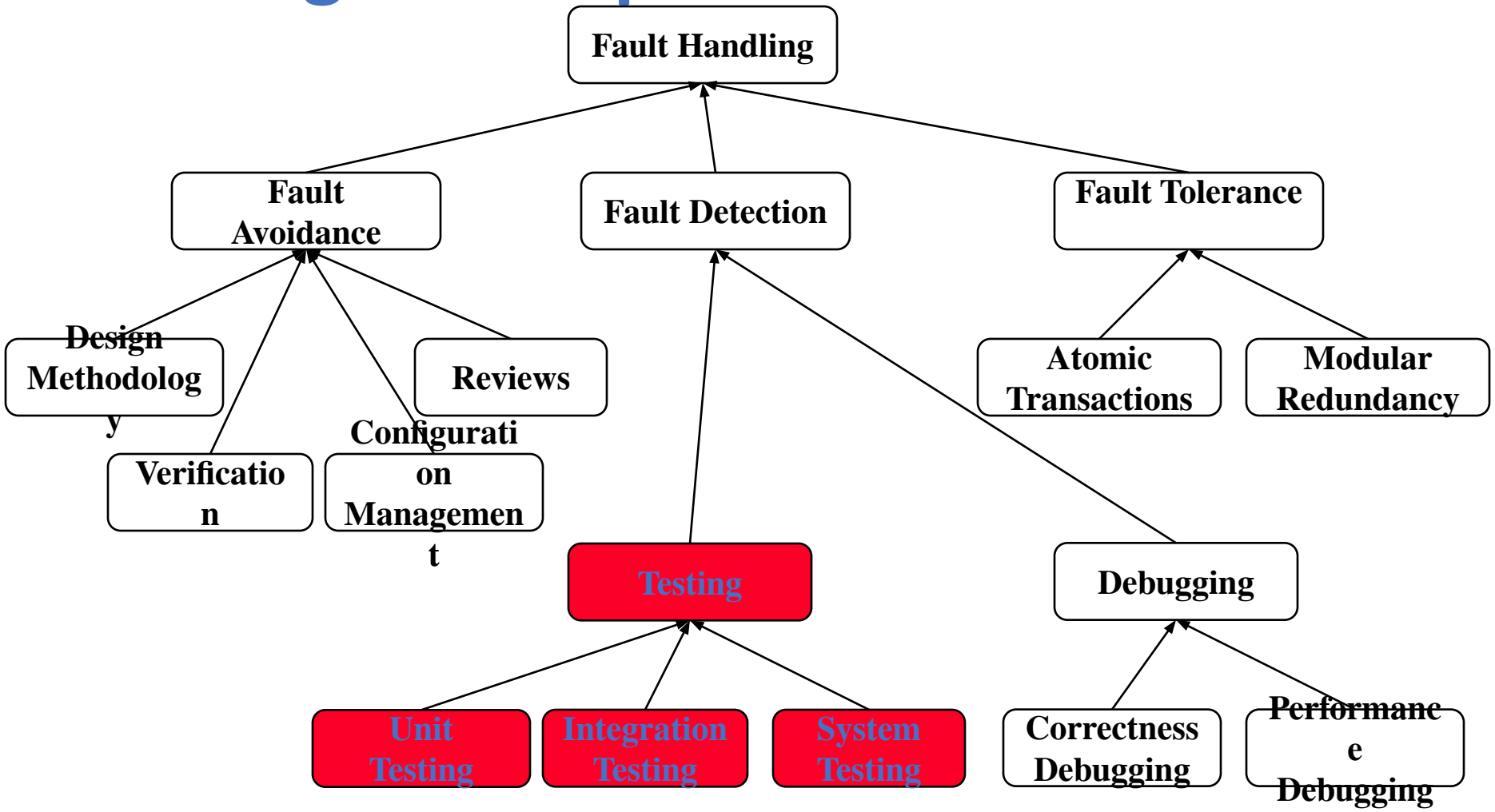
Software Quality Assurance Fundamentals

Some Observations

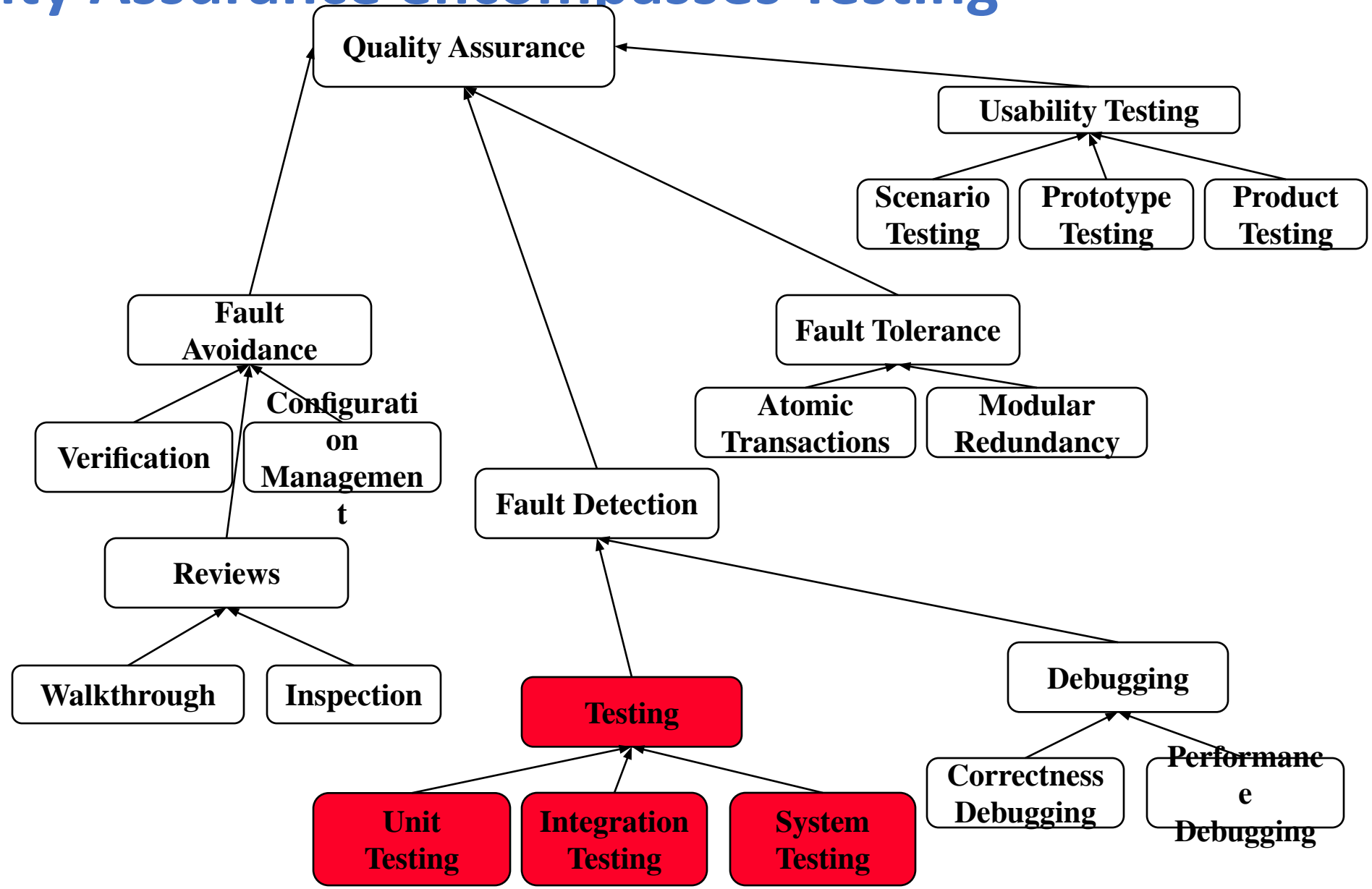
- It is impossible to completely test any nontrivial module or any system
  - Theoretical limitations: Halting prob
  - Practical limitations: Prohibitive in time and cost
- Testing can only show the presence of bugs, not their absence (Dijkstra)



Fault Handling Techniques



Quality Assurance encompasses Testing



Software Quality Assurance (SQA)

- It is the ongoing process that ensures the software product meets and complies with the organization's established and standardized quality specifications (quality factors, quality metrics)
- SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

Software Quality Assurance Plan

Abbreviated as SQAP, the software quality assurance plan comprises the procedures, techniques, and tools that are employed to make sure that a product or service aligns with the requirements defined in the SRS(software requirement specification).

SQA Activities

#1) Creating an SQA Management Plan: The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project. Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have the right talent mix in your team.

#2) Setting the Checkpoints: The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

#3) Apply software Engineering Techniques: Applying some software engineering techniques aids a software designer in achieving high-quality specifications. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique). Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

#4) Executing Formal Technical Reviews: An FTR is done to evaluate the quality and design of the prototype. In this process, a meeting is conducted with the technical staff to discuss the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

#5) Having a Multi-Testing Strategy: By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

SQA Activities

#6) Enforcing Process Adherence: This activity insists on the need for process adherence during the software development process. The development process should also stick to the defined procedures.

#7) Controlling Change: In this activity, we use a mix of manual procedures and automated tools to have a mechanism for change control. By validating the change requests, evaluating the nature of change, and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

#8) Measure Change Impact: If any defect is reported by the QA team, then the concerned team fixes the defect. After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project. For this purpose, we use software quality metrics that allow managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC and initiate corrective action wherever required.

#9) Performing SQA Audits: The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process. It also checks whether whatever was reported by the team in the status reports was actually performed or not. This activity also exposes any non-compliance issues.

#10) Maintaining Records and Reports: It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

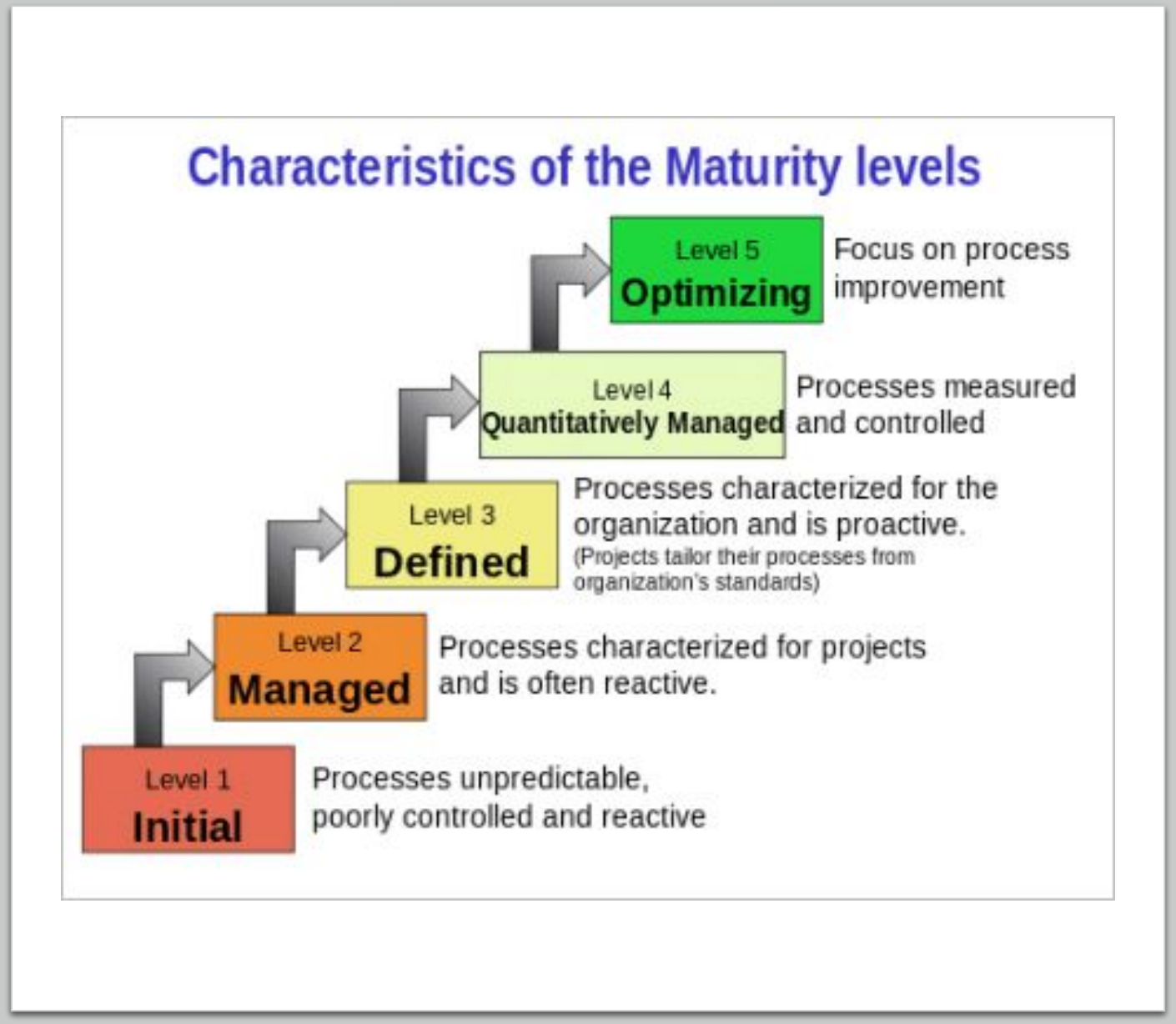
#11) Manage Good Relations: In fact, it is very important to maintain harmony between the QA and the development team. We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

Software Quality Assurance Standards

- **SO 9000:** This standard is based on seven quality management principles which help the organizations to ensure that their products or services are aligned with the customer needs.
- **7 principles of ISO 9000 are depicted in the below image:**

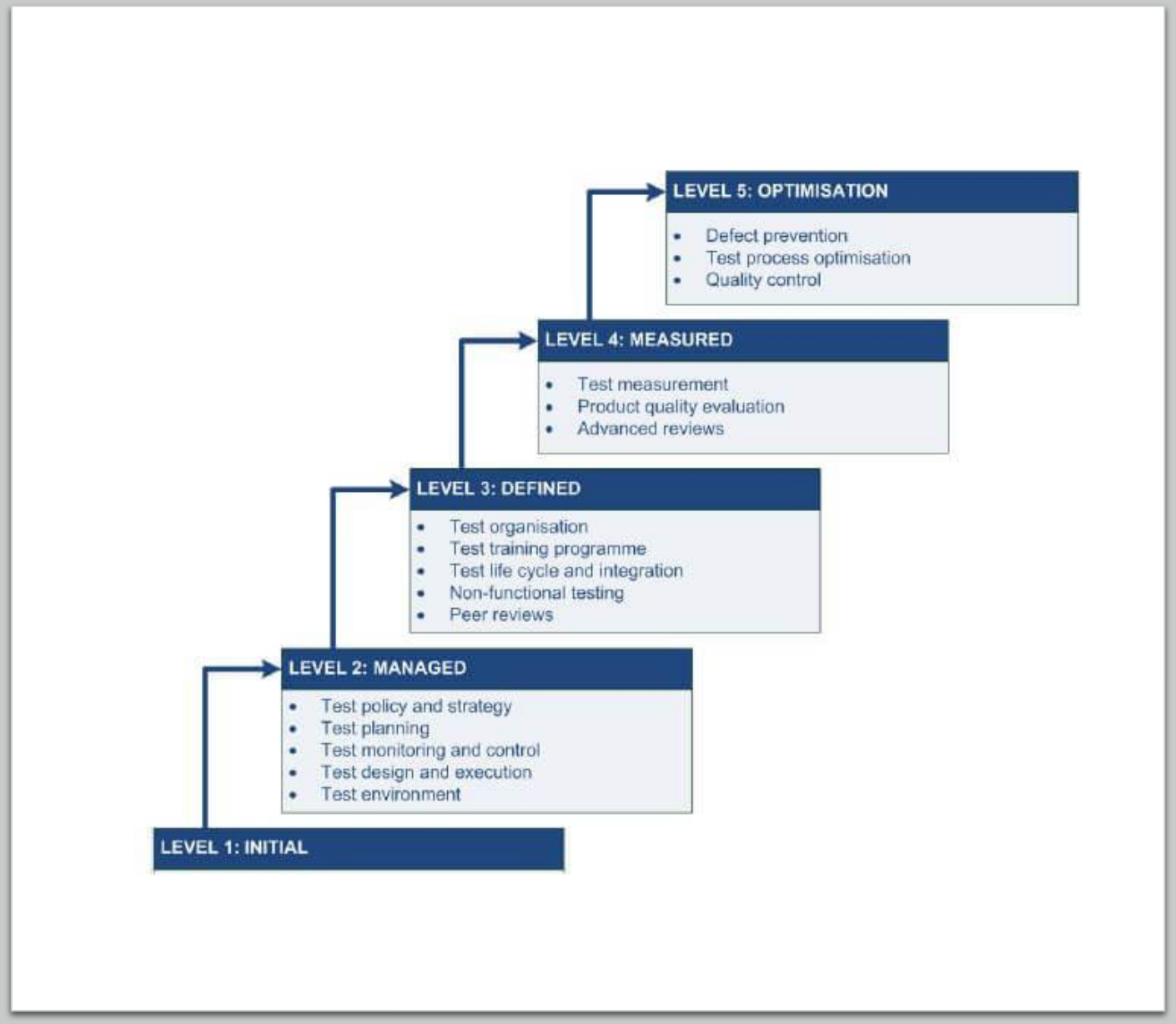
Software Quality Assurance Standards

- **CMMI level:** CMMI stands for **Capability maturity model Integration**. This model originated in software engineering. It can be employed to direct process improvement throughout a project, department, or entire organization.
- **5 CMMI levels and their characteristics are described in the below image:**



Software Quality Assurance Standards

- **Test Maturity Model integration (TMMi):** Based on CMMi, this model focuses on maturity levels in software quality management and testing.
- **5 TMMi levels are depicted in the below image:**



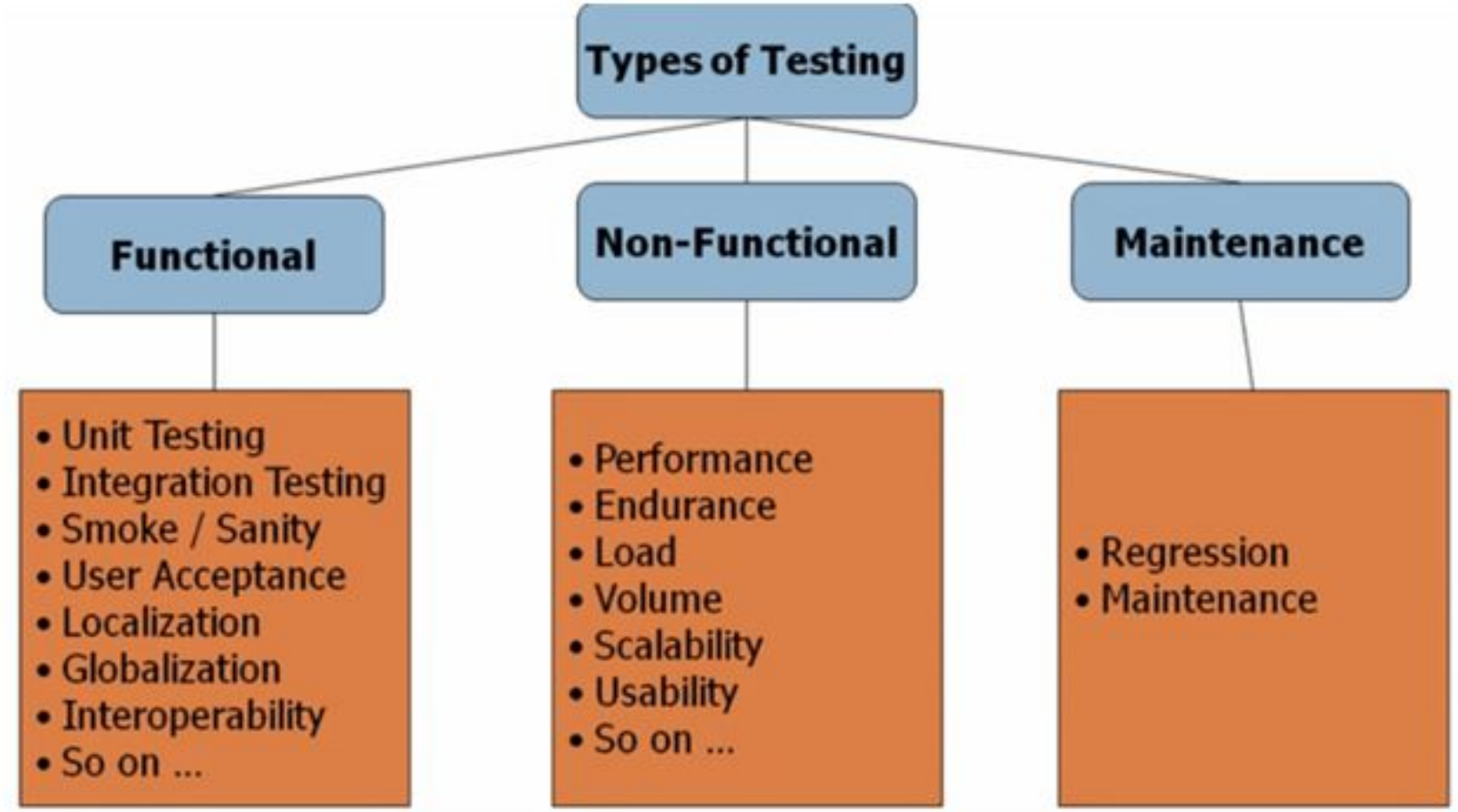
Elements of Software Quality Assurance

There are 10 essential elements of SQA which are enlisted below for your reference:

- Software engineering Standards
- Technical reviews and audits
- Software Testing for quality control
- Error collection and analysis
- Change management
- Educational programs
- Vendor management
- Security management
- Safety
- Risk management

Software Testing Types

Functional and Non-Functional testing



Non-Functional testing

This testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

Non-Functional testing

Test Case #	Test Case	Domain
1	Application load time should not be more than 5 secs up to 1000 users accessing it simultaneously	Performance Testing
2	Software should be installable on all versions of Windows and Mac	Compatibility Testing
3	All web images should have alt tags	Accessibility testing.

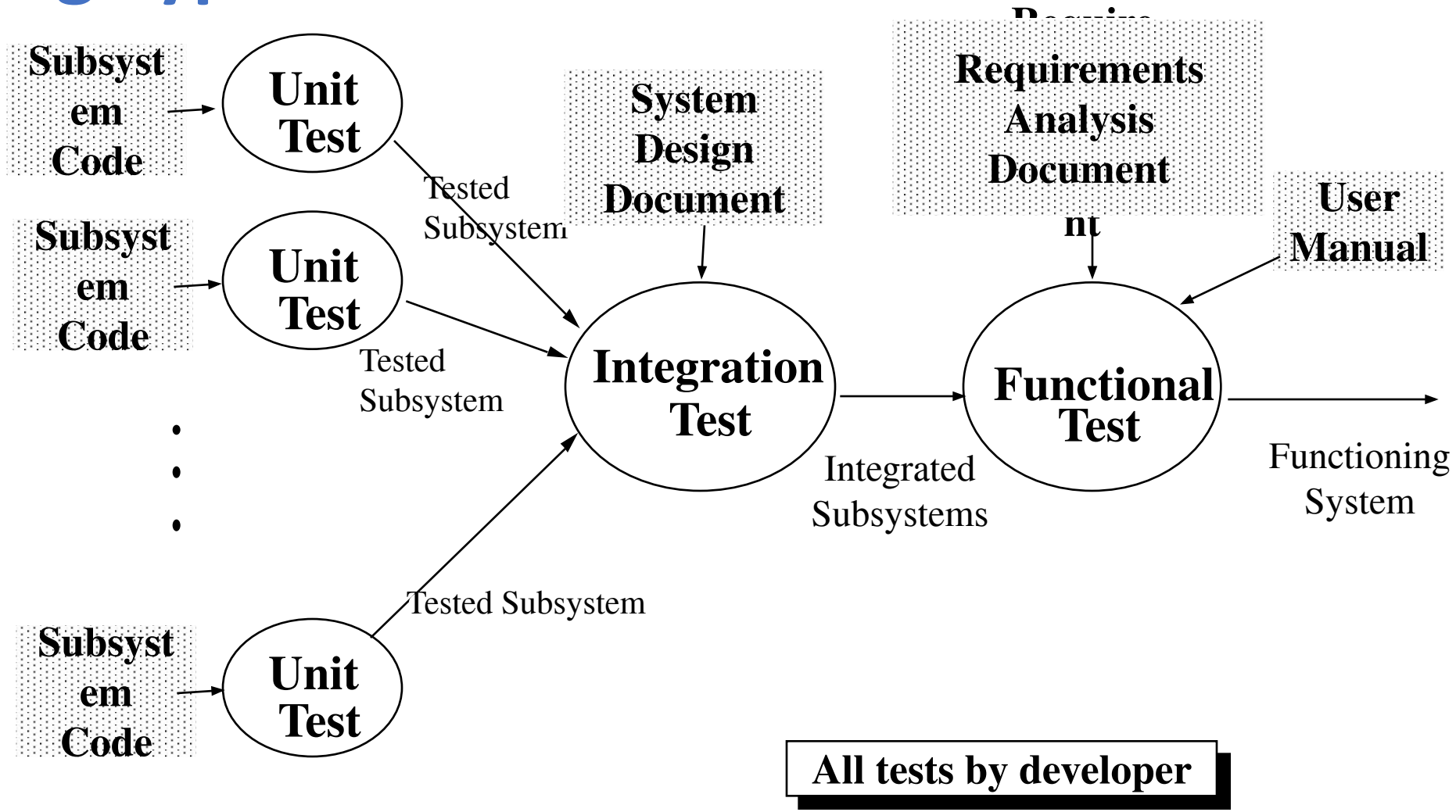
Functional Testing

- It is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.
- Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.

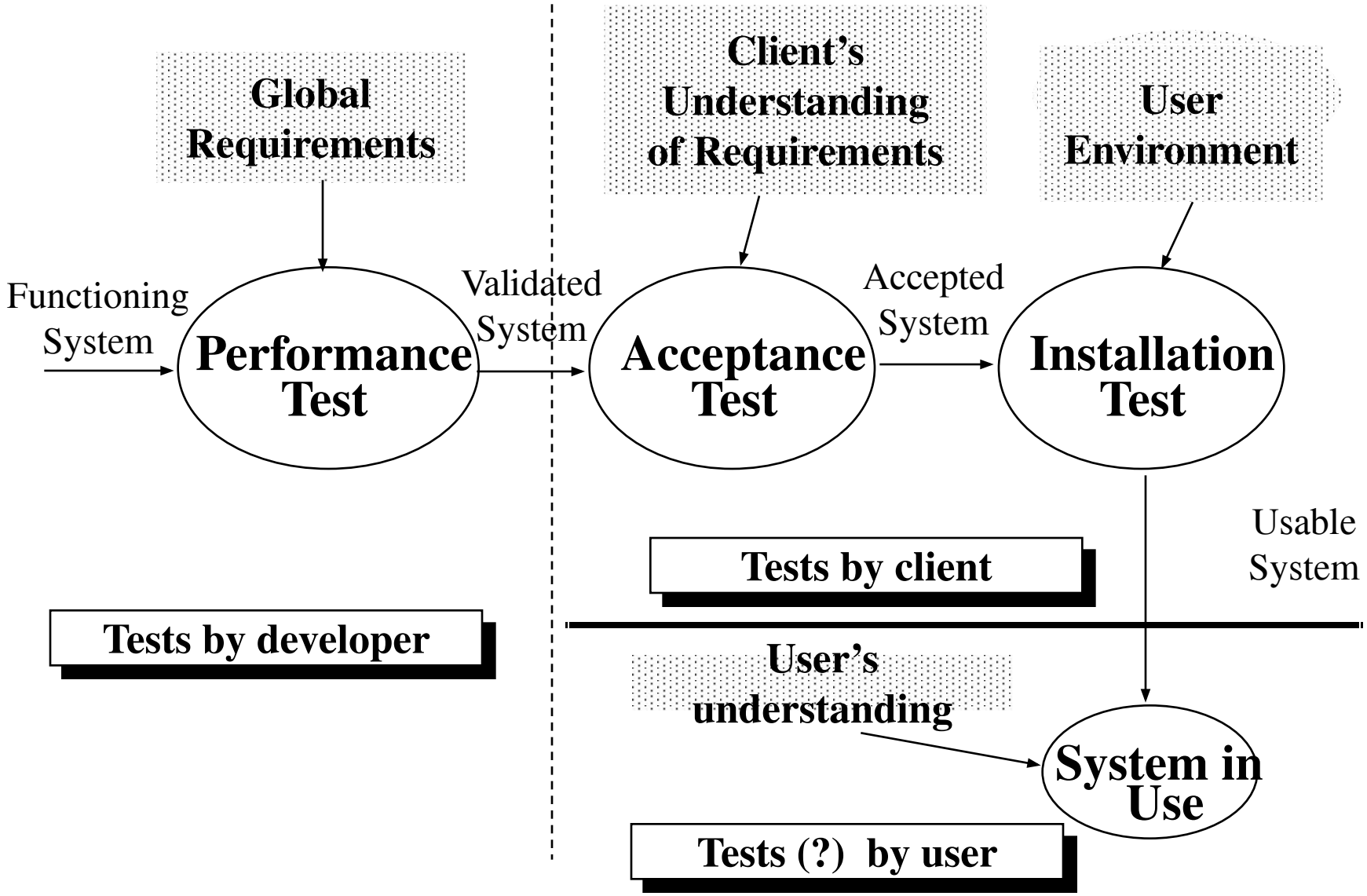
Functional Vs Non-Functional Testing

Functional Testing	Non-Functional Testing
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.	Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Non-functional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing.
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing
• Unit Testing, Smoke Testing, Sanity Testing, Integration Testing, White box testing, Black Box testing, User Acceptance testing, Regression Testing	Performance Testing, Load Testing, Volume Testing, Stress Testing, Security Testing, Installation Testing, Penetration Testing, Compatibility Testing, Migration Testing

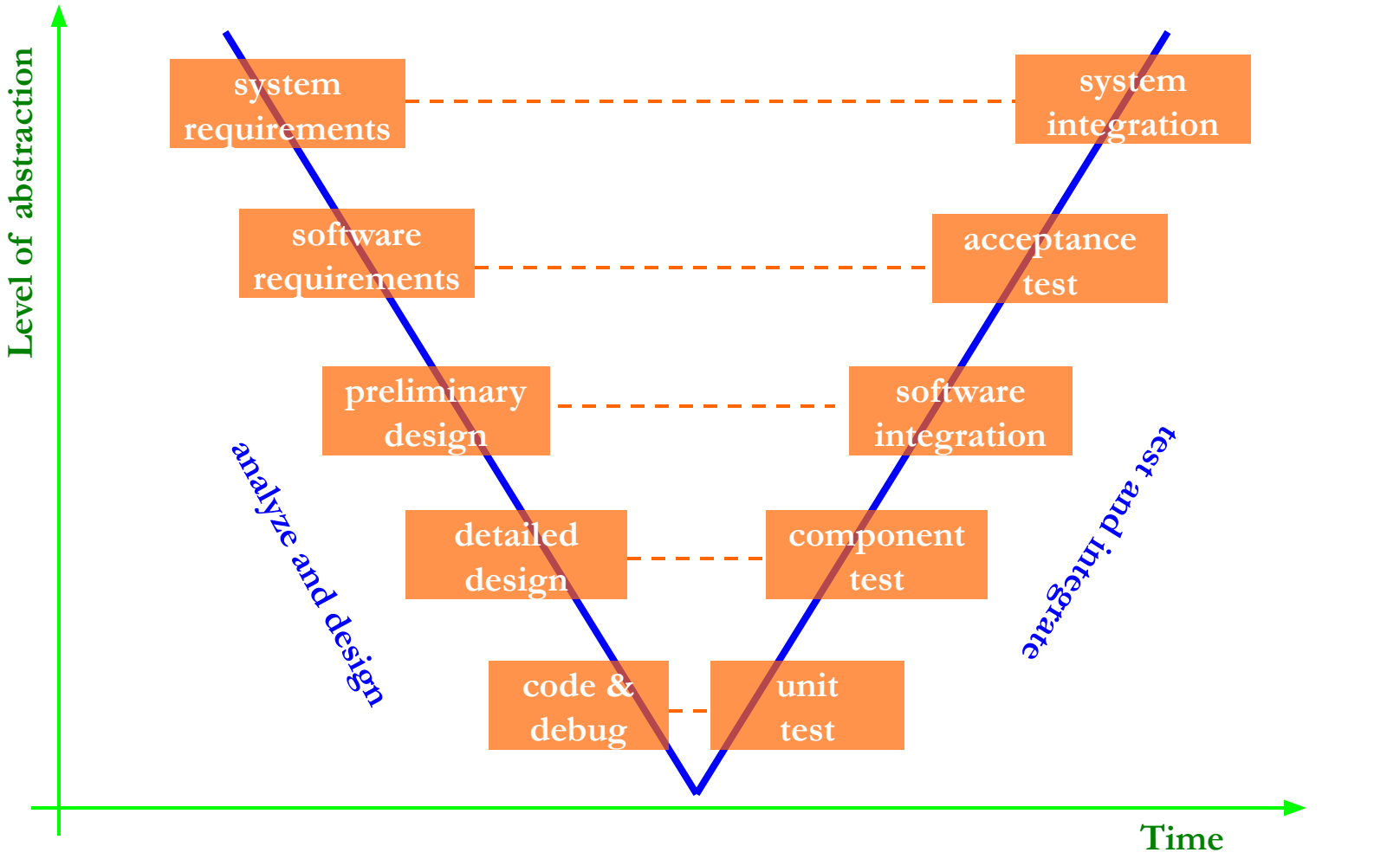
Testing Types



Testing Types continued



Levels of Testing in V Model



N.B.: component test vs. unit test; acceptance test vs. system integration

Types of Testing

- **Unit** Testing:
  - Individual *subsystem*
  - Carried out by developers
  - Goal: Confirm that subsystems is correctly coded and carries out the intended functionality
- **Integration** Testing:
  - Groups of subsystems (collection of classes) and eventually the entire system
  - Carried out by developers
  - Goal: Test the *interface* among the subsystem

Types of Testing

- **System** Testing:
  - The entire system
  - Carried out by developers
  - Goal: Determine if the system meets the *requirements* (functional and *global*)
- **Acceptance** Testing:
  - Evaluates the system delivered by developers
  - Carried out by the *client*. May involve executing typical transactions on site on a trial basis
  - Goal: Demonstrate that the system meets customer *requirement*s and is ready to use

2 kinds of Acceptance testing

Unit Testing

- Informal:
  - Incremental coding **Write a little, test a little**
- Static Analysis:
  - Hand execution: Reading the *source code*
  - Walk-Through (informal presentation to others)
  - Code Inspection (formal presentation to others)
  - Automated Tools checking for
    - syntactic and semantic errors
    - departure from coding standards
- Dynamic Analysis:
  - Black-box testing (Test the input/output behavior)
  - *White-box* testing (Test the internal logic of the subsystem or object)
  - Data-structure based testing (Data types determine test cases)

Which is more effective, static or dynamic analysis?

Black-Box vs. White-Box Testing

Black-box Testing

- Focus: I/O behavior. If for any given input, we can predict the output, then the module passes the test.
  - Almost always impossible to generate all possible inputs ("test cases")

why?

- Goal: Reduce number of test cases by equivalence partitioning:
  - Divide input conditions into equivalence classes
  - Choose test cases for each equivalence class. (Example: If an object is supposed to accept a negative number, testing one negative number is enough)

❑ **If x = 3 then …**

❑ **If x > -5 and x < 5 then …**

What would be the equivalence classes?

Black-box Testing (Continued)

- Selection of equivalence classes (***No*** rules, only guidelines):
  - Input is valid across range of values. Select test cases from 3 equivalence classes:
    - Below the range
    - Within the range
    - Above the range

    Are these complete?
  - Input is valid if it is from a discrete set. Select test cases from 2 equivalence classes:
    - Valid discrete value
    - Invalid discrete value
- Another solution to select only a limited amount of test cases:
  - Get knowledge about the inner workings of the unit being tested => white-box testing

White-box Testing

- Focus: Thoroughness (Coverage). Every statement in the component is executed at least once.
- Four types of white-box testing
  - Statement Testing
  - Loop Testing
  - Path Testing
  - Branch Testing

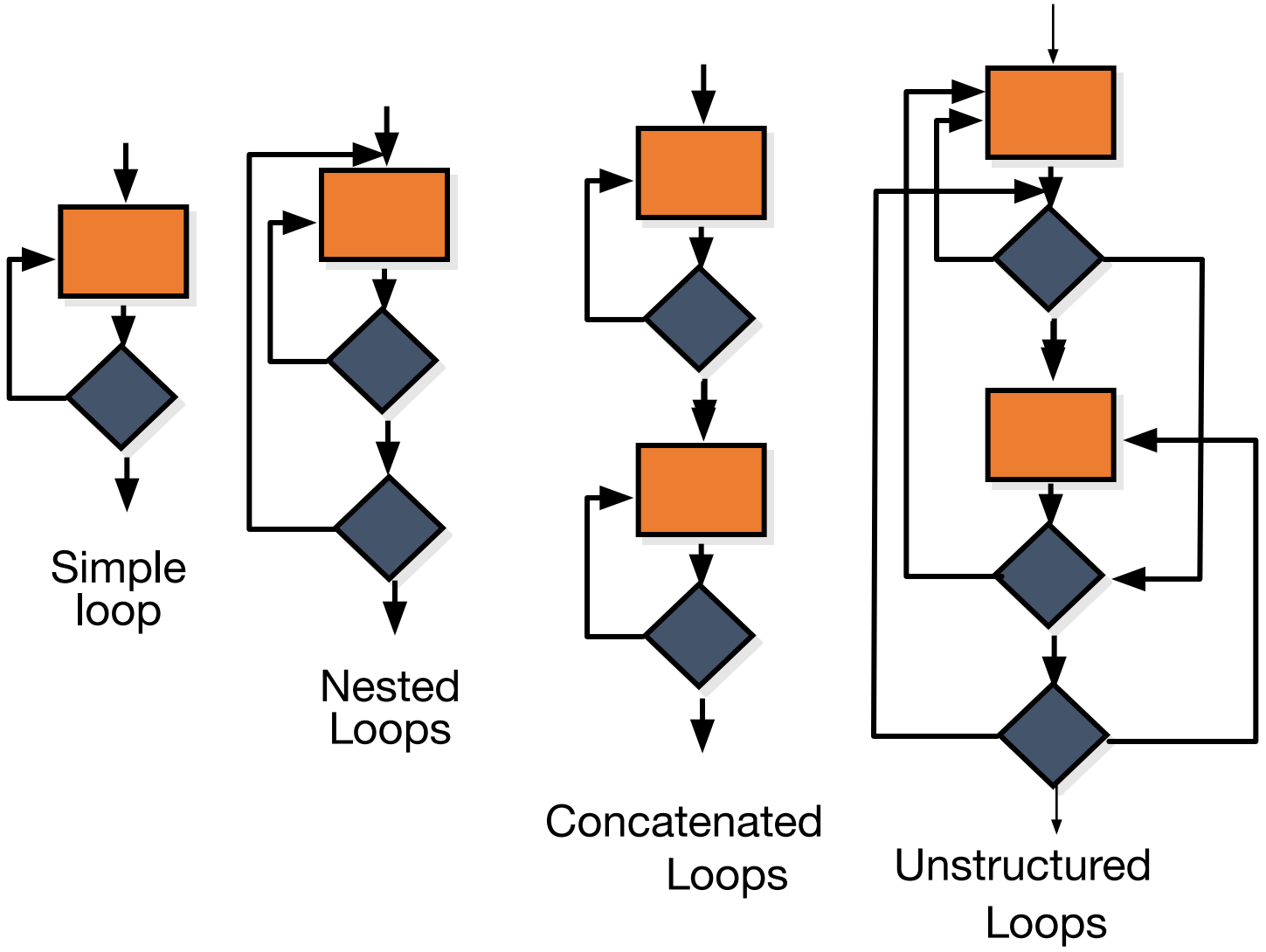
White-box Testing (Continued)

- Statement Testing (Algebraic Testing): Test single statements
- Loop Testing:
  - Cause execution of the loop to be skipped completely. (Exception: Repeat loops)
  - Loop to be executed exactly once
  - Loop to be executed more than once
- Path testing:
  - Make sure all paths in the program are executed
- Branch Testing (Conditional Testing): Make sure that each possible outcome from a condition is tested at least once

```
if ( i =TRUE) printf("YES\n");else printf("NO\n");
Test cases: 1) i = TRUE; 2) i = FALSE
```

White-Box Testing: Loop Testing

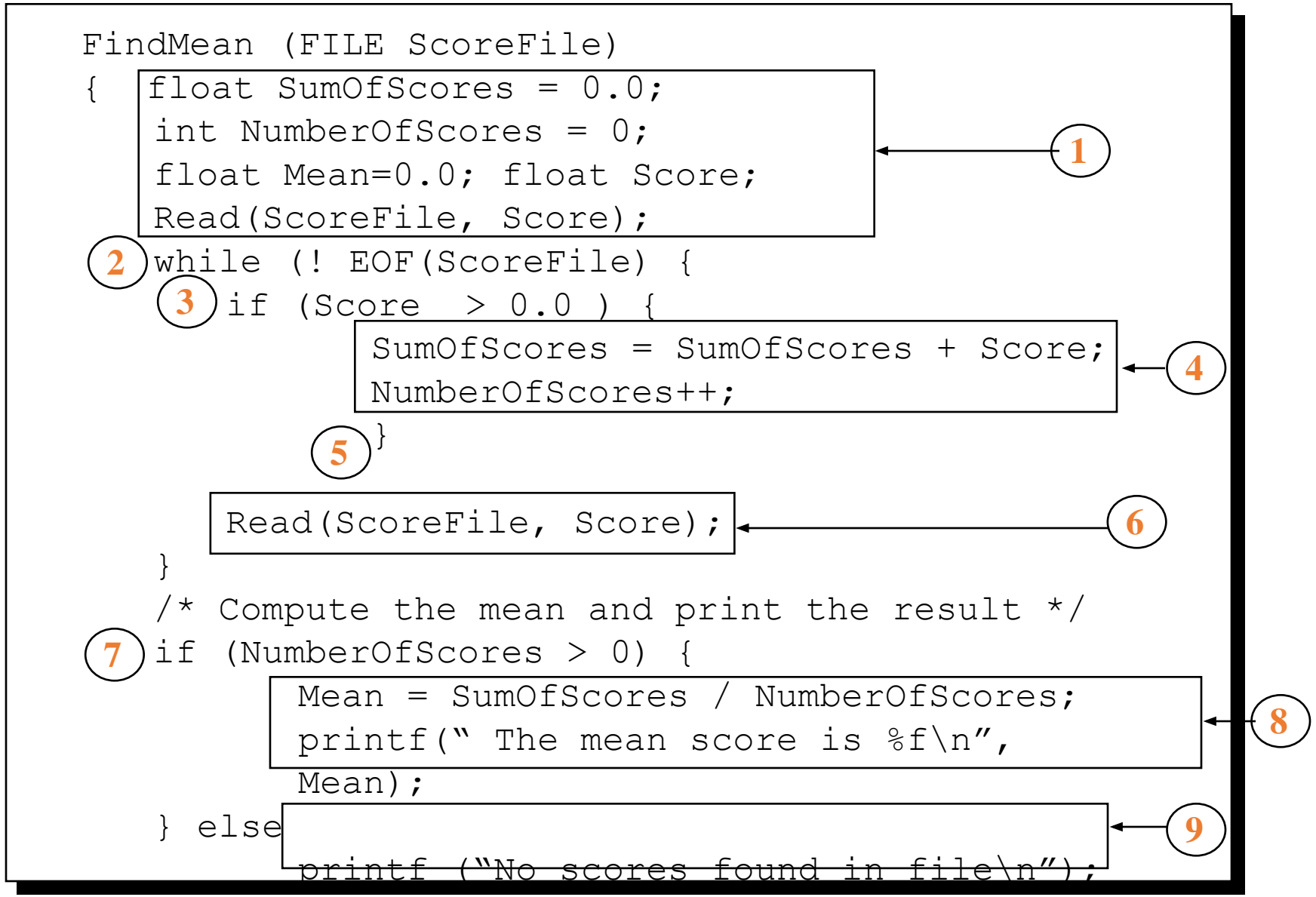
[Pressman]



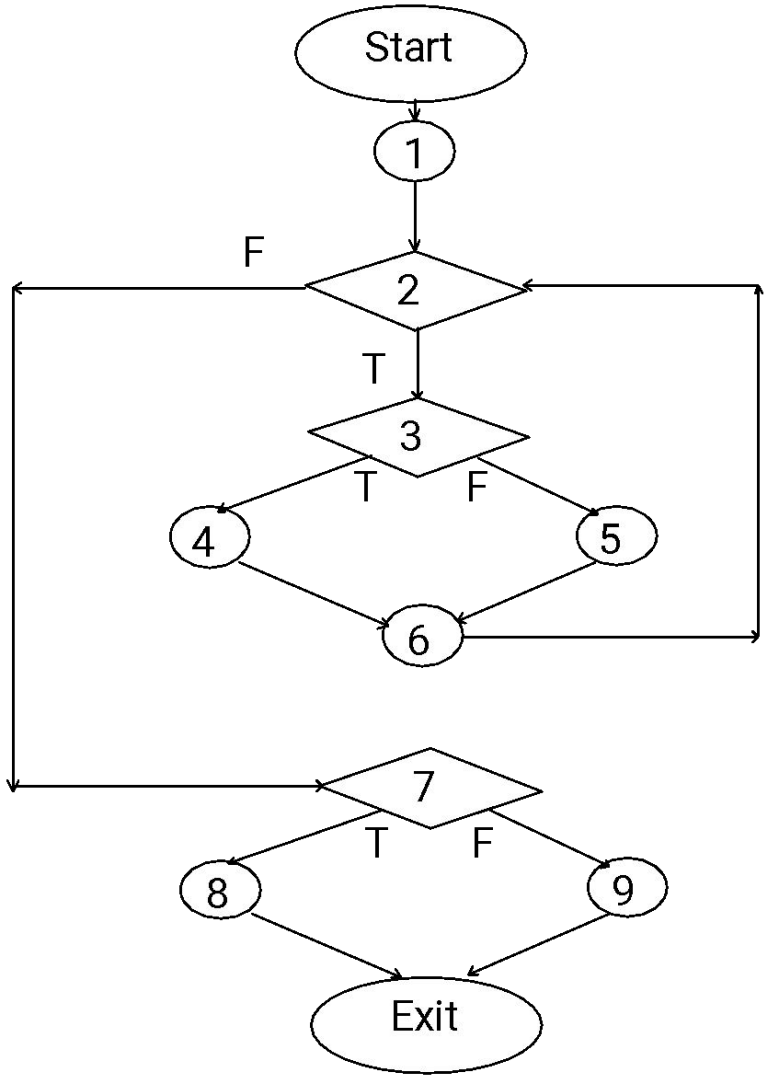
White-box Testing Example

```
FindMean(float Mean, FILE ScoreFile)
{ SumOfScores = 0.0; NumberOfScores = 0; Mean =
0;
 Read(ScoreFile, Score);  /*Read in and sum the
 while (! EOF(ScoreFile) {scores*/
         if ( Score > 0.0 )
        {       SumOfScores = SumOfScores +
                Score;
          }     NumberOfScores++;
          Read(ScoreFile,
 }       Score);
 /* Compute the mean and print the result
*/if (NumberOfScores > 0 ) {
              Mean =
              SumOfScores/NumberOfScores;
              printf("The mean score is %f \n",
 } else       Mean);
              printf("No scores found in
              file\n");
}
```

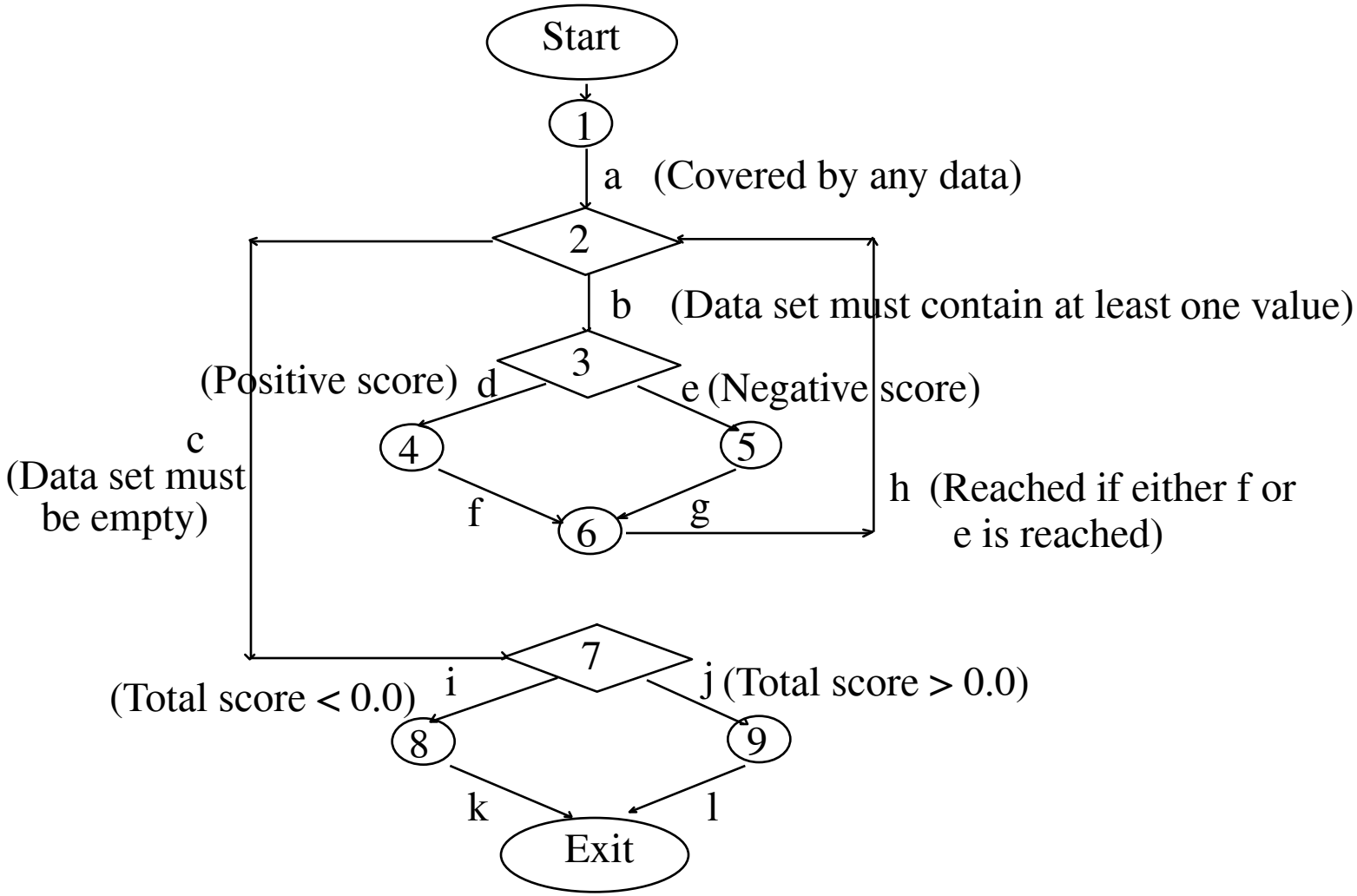
White-box Testing Example: Determining the Paths



Constructing the Logic Flow Diagram



Finding the Test Cases

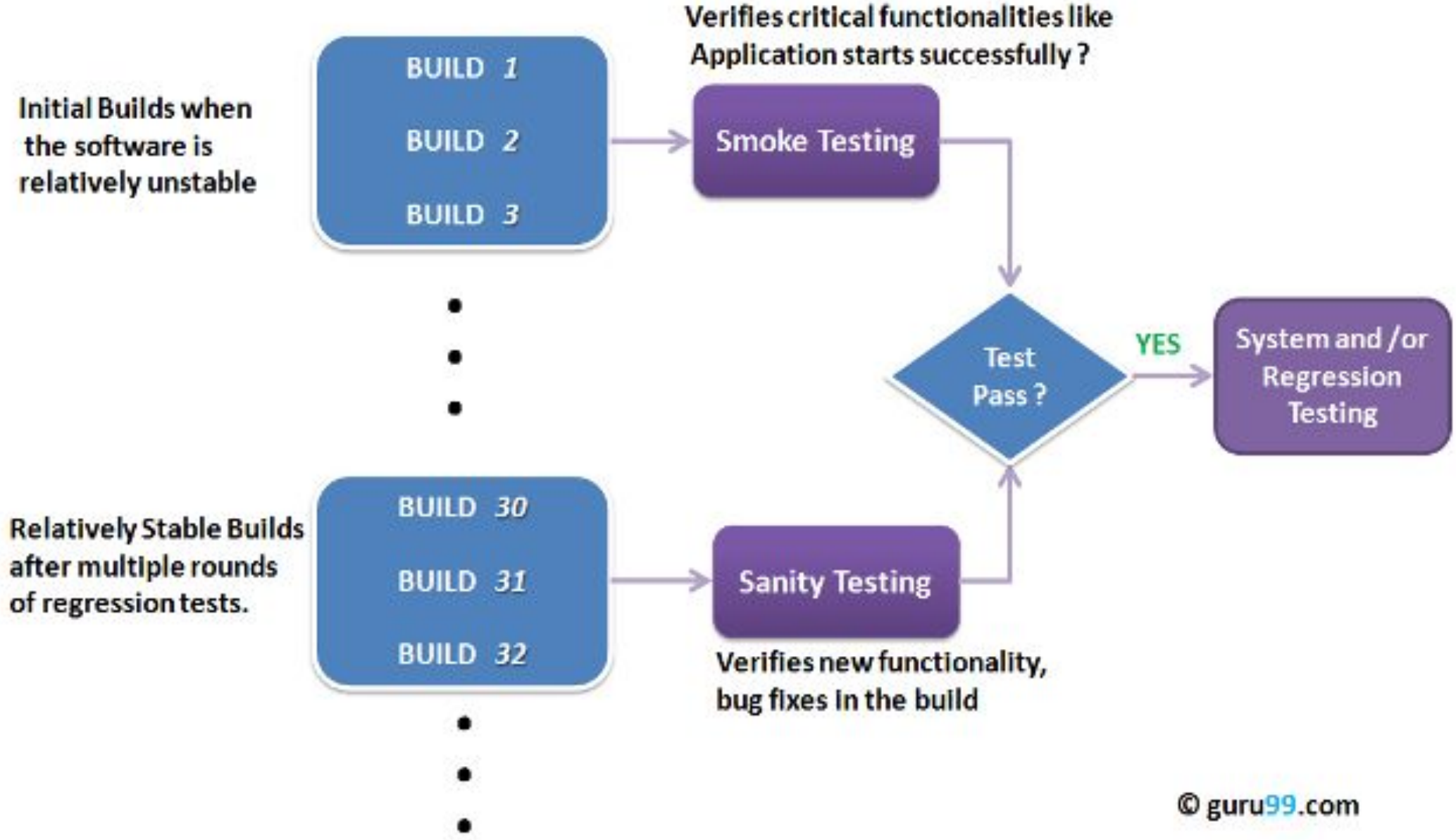


Comparison of White & Black-box Testing

- White-box Testing:
  - Potentially infinite number of paths have to be tested
  - White-box testing often tests what is done, instead of what should be done
  - Cannot detect missing use cases
- Black-box Testing:
  - Potential combinatorical explosion of test cases (valid & invalid data)
  - Often not clear whether the selected test cases uncover a particular error
  - Does not discover extraneous use cases ("features")

- Both types of testing are needed
- White-box testing and black box testing are the extreme ends of a testing continuum.
- Any choice of test case lies in between and depends on the following:
  - Number of possible logical paths
  - Nature of input data
  - Amount of computation
  - Complexity of algorithms and data structures

Sanity vs. Smoke Testing



Smoke Testing vs Sanity Testing

What is a Software Build?

If you are developing a simple computer program which consists of only one source code file, you merely need to compile and link this one file, to produce an executable file. This process is very simple.
Usually, this is not the case. A typical Software Project consists of hundreds or even thousands of source code files. Creating an executable program from these source files is a complicated and time-consuming task.

You need to use “build” software to create an executable program and the process is called ” *Software Build*”

Smoke Testing

- **Smoke Testing** is a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.
- In smoke testing, the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.
  For Example, a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive … etc.

Sanity Testing

- Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.
- The objective is “not” to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

KEY DIFFERENCE

- Smoke Testing has a goal to verify “stability” whereas Sanity Testing has a goal to verify “rationality”.
- Smoke Testing is done by both developers or testers whereas Sanity Testing is done by testers.
- Smoke Testing verifies the critical functionalities of the system whereas Sanity Testing verifies the new functionality like bug fixes.
- Smoke testing is a subset of acceptance testing whereas Sanity testing is a subset of Regression Testing.
- Smoke testing is documented or scripted whereas Sanity testing isn’t.
- Smoke testing verifies the entire system from end to end whereas Sanity Testing verifies only a particular component.

Smoke Testing	Sanity Testing
Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine	Sanity Testing is done to check the new functionality/bugs have been fixed
The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing	The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing
This testing is performed by the developers or testers	Sanity testing in software testing is usually performed by testers
Smoke testing is usually documented or scripted	Sanity testing is usually not documented and is unscripted
Smoke testing is a subset of Acceptance testing	Sanity testing is a subset of Regression Testing
Smoke testing exercises the entire system from end to end	Sanity testing exercises only the particular component of the entire system
Smoke testing is like General Health Check Up	Sanity Testing is like specialized health check up

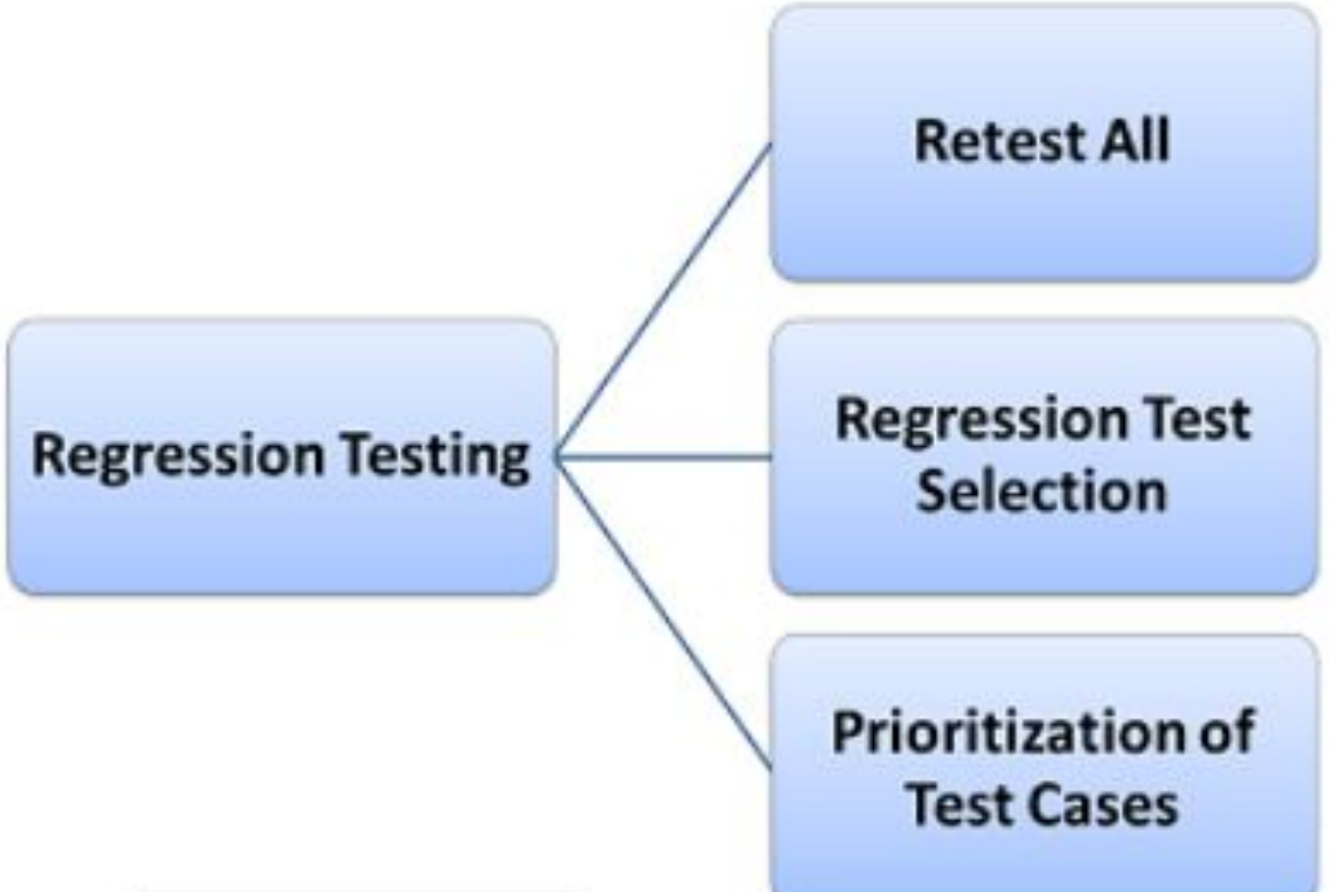
Points to note

- Both Sanity and Smoke testing are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing.
- Smoke Testing is also called tester acceptance testing.
- Smoke testing performed on a particular build is also known as a build verification test.
- One of the best industry practice is to conduct a Daily build and smoke test in software projects.
- Both smoke and sanity tests can be executed manually or using an automation tool. When automated tools are used, the tests are often initiated by the same process that generates the build itself.
- As per the needs of testing, you may have to execute both Sanity and Smoke Tests in the software build. In such cases, you will first execute Smoke tests and then go ahead with Sanity Testing. In industry, test cases for Sanity Testing are commonly combined with that for smoke tests, to speed up test execution. Hence, it's a common that the terms are often confused and used interchangeably

What is Regression Testing?

- **Regression Testing** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Regression Testing



Selecting test cases for regression testing

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

Difference between Re-Testing and Regression Testing

- **Retesting** is a process to check specific test cases that are found with bug/s in the final execution. Generally, testers find these bugs while testing the software application and assign it to the developers to fix it. Then the developers fix the bug/s and assign it back to the testers for verification
- Regression testing is performed for passed test cases while Retesting is done only for failed test cases.
- Regression testing checks for unexpected side-effects while Re-testing makes sure that the original fault has been corrected.
- Regression Testing doesn’t include defect verification whereas Re-testing includes defect verification.
- Regression testing is known as generic testing whereas Re-testing is planned testing.
- Regression Testing is possible with the use of automation whereas Re-testing is not possible with automation.

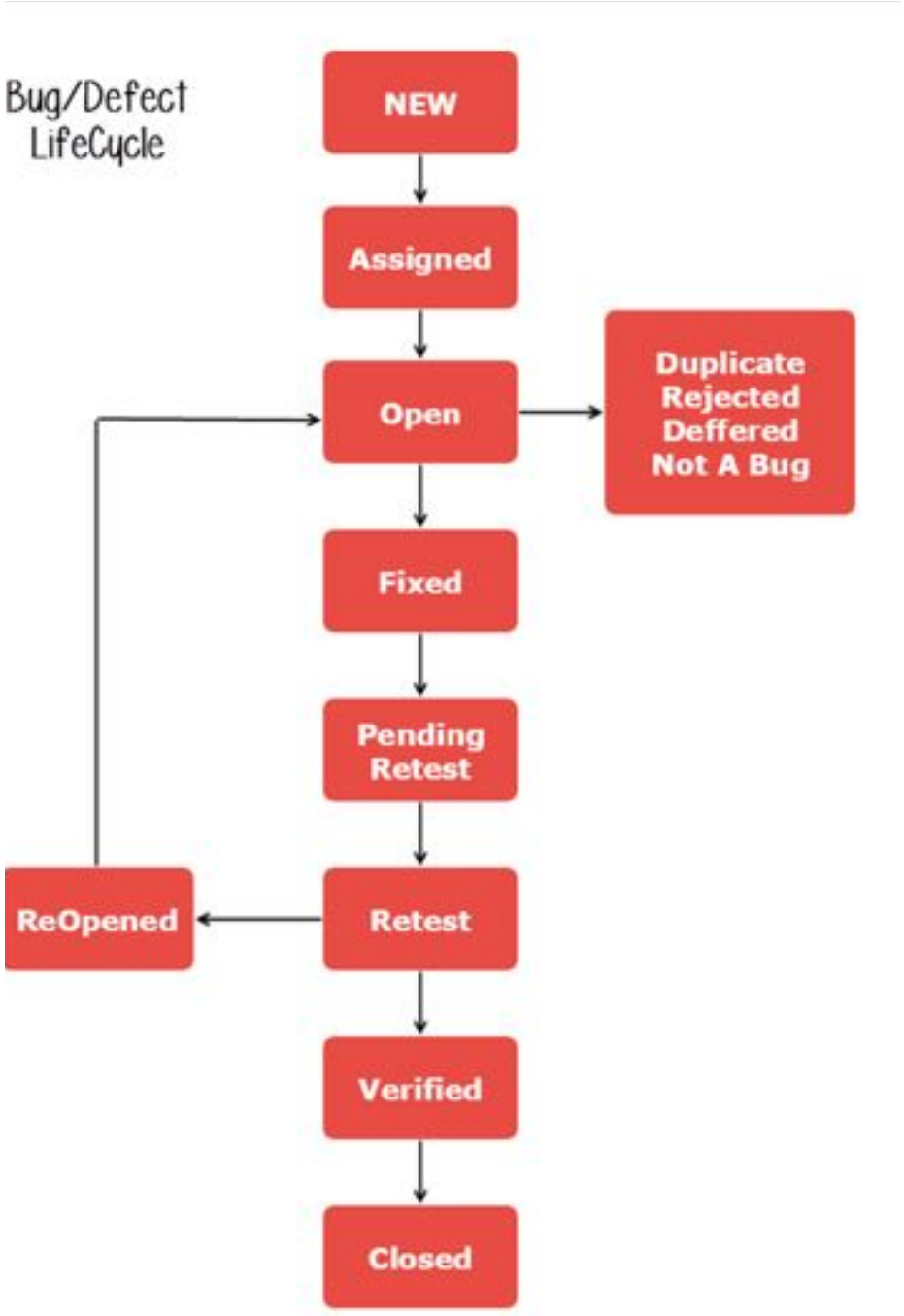
Regression Testing	Re-testing
•Regression Testing is carried out to confirm whether a recent program or code change has not adversely affected existing features	•Re-testing is carried out to confirm the test cases that failed in the final execution are passing after the defects are fixed
•The purpose of Regression Testing is that new code changes should not have any side effects to existing functionalities	•Re-testing is done on the basis of the Defect fixes
•Defect verification is not the part of Regression Testing	•Defect verification is the part of re-testing
•Based on the project and availability of resources, Regression Testing can be carried out parallel with Re-testing	•Priority of re-testing is higher than regression testing, so it is carried out before regression testing
•You can do automation for regression testing, Manual Testing could be expensive and time-consuming	•You cannot automate the test cases for Retesting
•Regression testing is done for passed test cases	•Retesting is done only for failed test cases
•Regression testing checks for unexpected side-effects	•Re-testing makes sure that the original fault has been corrected
•Regression testing is only done when there is any modification or changes become mandatory in an existing project	•Re-testing executes a defect with the same data and the same environment with different inputs with a new build
•Test cases for regression testing can be obtained from the functional specification, user tutorials and manuals, and defect reports in regards to corrected problems	•Test cases for retesting cannot be obtained before start testing.

Bug definition

- A bug is the consequence/outcome of a coding fault.
- A **Defect in Software Testing** is a variation or deviation of the software application from end user's requirements or original business requirements. A software defect is an error in coding which causes incorrect or unexpected results from a software program which does not meet actual requirements. Testers might come across such defects while executing the test cases.
- These two terms have very thin line of difference, In the Industry both are faults that need to be fixed and so interchangeably used by some of the testing teams.
- When testers execute the test cases, they might come across such test results which are contradictory to expected results. This variation in test results is referred to as a Software Defect. These defects or variations are referred by different names in different organizations like issues, problems, bugs or incidents.

Defect/Bug Life Cycle in Software Testing

1. Tester finds the defect
2. Status assigned to defect- New
3. A defect is forwarded to Project Manager for analyze
4. Project Manager decides whether a defect is valid
5. Here the defect is not valid- a status is given “Rejected.”
6. So, project manager assigns a status **rejected**. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a **postponed or deferred** status.
7. Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status **duplicate**.
8. If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status **in- progress.**
9. Once the code is fixed. A defect is assigned a status **fixed**
10. Next, the tester will re-test the code. In case, the Test case passes the defect is **closed.** If the test cases fail again, the defect is **re-opened** and assigned to the developer.
11. Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be **re-opened.**



Test Documentation: a Bug Report

Defect_ID – Unique identification number for the defect.

Defect Description – Detailed description of the Defect including information about the module in which Defect was found.

Version – Version of the application in which defect was found.

Steps – Detailed steps along with screenshots with which the developer can reproduce the defects.

Date Raised – Date when the defect is raised

Reference– where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect

Detected By – Name/ID of the tester who raised the defect

Status – Status of the defect , more on this later

Fixed by – Name/ID of the developer who fixed it

Date Closed – Date when the defect is closed

Severity which describes the impact of the defect on the application

Priority which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively

Test Planning

- A Test Plan:
  - covers all types and phases of testing
  - guides the entire testing process
  - who, why, when, what
  - developed as requirements, functional specification, and high-level design are developed
  - should be done before implementation starts

- A test plan includes:
  - **test objectives**
  - **schedule and logistics**
  - **test strategies**
  - **test cases**
    - **procedure**
    - **data**
    - **expected result**
  - **procedures for handling problems**

Terminology

- **Reliability:** The measure of success with which the observed behavior of a system confirms to some specification of its behavior.
- **Failure:** Any deviation of the observed behavior from the specified behavior.
- **Error:** The system is in a state such that further processing by the system will lead to a failure
- **Fault (Bug):** The mechanical or algorithmic cause of an error.

Examples of Faults and Errors

- Faults in the Interface specification
  - Mismatch between what the client needs and what the server offers
  - Mismatch between requirements and implementation
- Algorithmic Faults
  - Missing initialization
  - Branching errors (too soon, too late)
  - Missing test for nil

- Mechanical Faults (very hard to find)
  - Documentation does not match actual conditions or operating procedures
- Errors
  - Stress or overload errors
  - Capacity or boundary errors
  - Timing errors
  - Throughput or performance errors

Who Tests the Software?

developer

Understands the system

but, will test "gently"

and, is driven by "*delivery*"

independent tester

Must learn about the system,

but, will attempt to **break** it

and, is driven by *quality*

Balzhan Azibek (Балжан Бекбайқызы)

balzhan.azibek@astanait.edu.kz

www.astanait.edu.kz