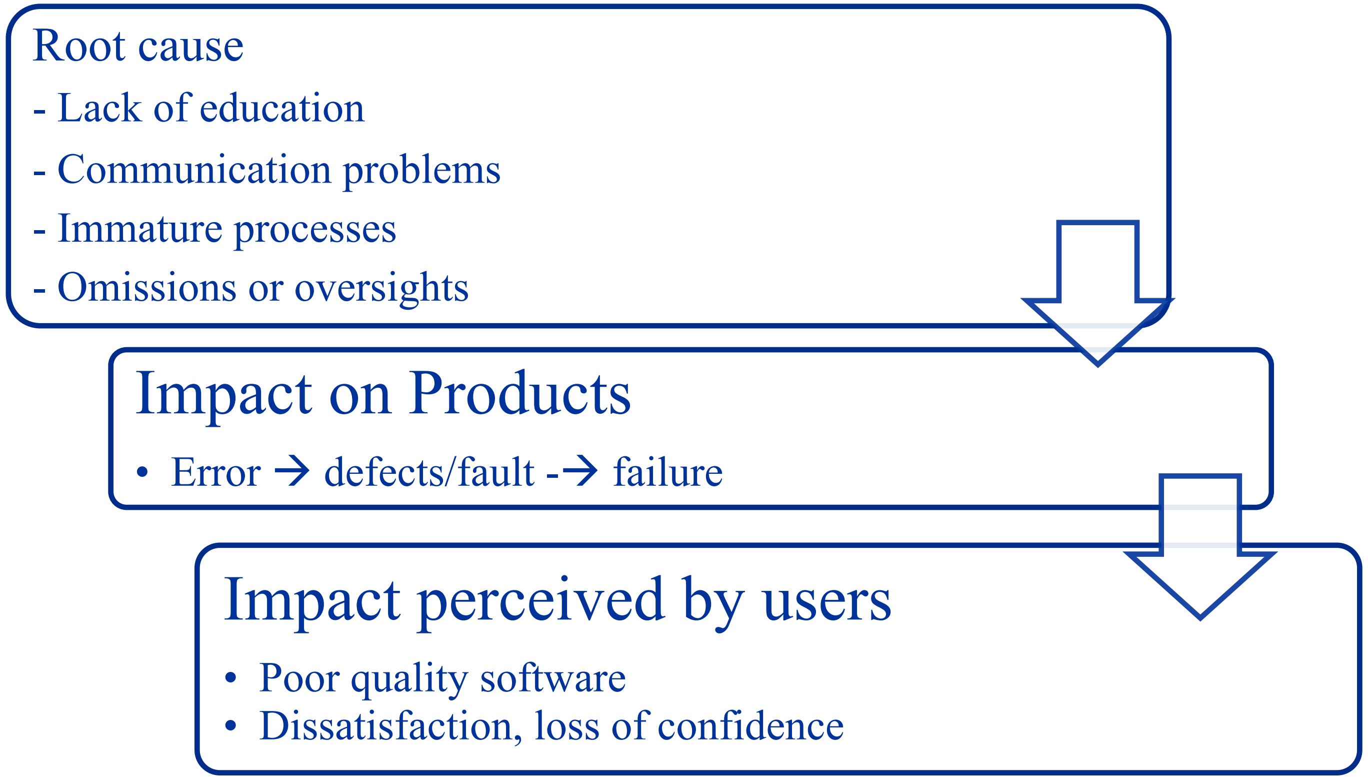
Lecture 17: Testing

Testing

- *Testing* a set of activities with the objective of identifying failures in a software or system and to evaluate its level of quality, to obtain user satisfaction. It is a set of tasks with clearly defined goals.

Defects

- Defects are introduced at the same time the code is written, by the same persons.



Root cause
- Lack of education
- Communication problems
- Immature processes
- Omissions or oversights

Impact on Products
- Error → defects/fault -→ failure

Impact perceived by users
- Poor quality software
- Dissatisfaction, loss of confidence

Sources of Problems

- Requirements Definition: Erroneous, incomplete, inconsistent requirements.
- Design: Fundamental design flaws in the software.
- Implementation: Mistakes in chip fabrication, wiring, programming faults, malicious code.
- Support Systems: Poor programming languages, faulty compilers and debuggers, misleading development tools.

Sources of Problems (2)

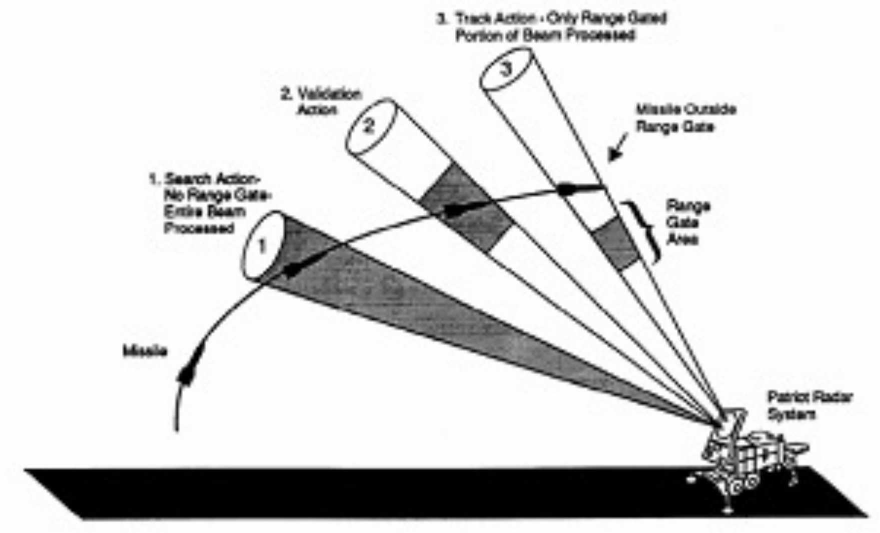
- Inadequate Testing of Software: Incomplete testing, poor verification, mistakes in debugging.
- Evolution: Sloppy redevelopment or maintenance, introduction of new flaws in attempts to fix old flaws, incremental escalation to inordinate complexity.

Common Goals of Testing

- Software testing focuses on two complementary but distinct aspects:
  - defect and failure detection, so that these can be fixed and thus the quality of the product delivered to customers and users is improved;
  - allows decisions to be made on the basis of the information provided regarding the level of risks associated with the delivery of the software to the market, and on the efficiency of the organization's processes which are the root cause of the identified defects and/or failures.

Patriot Missile Software Problem

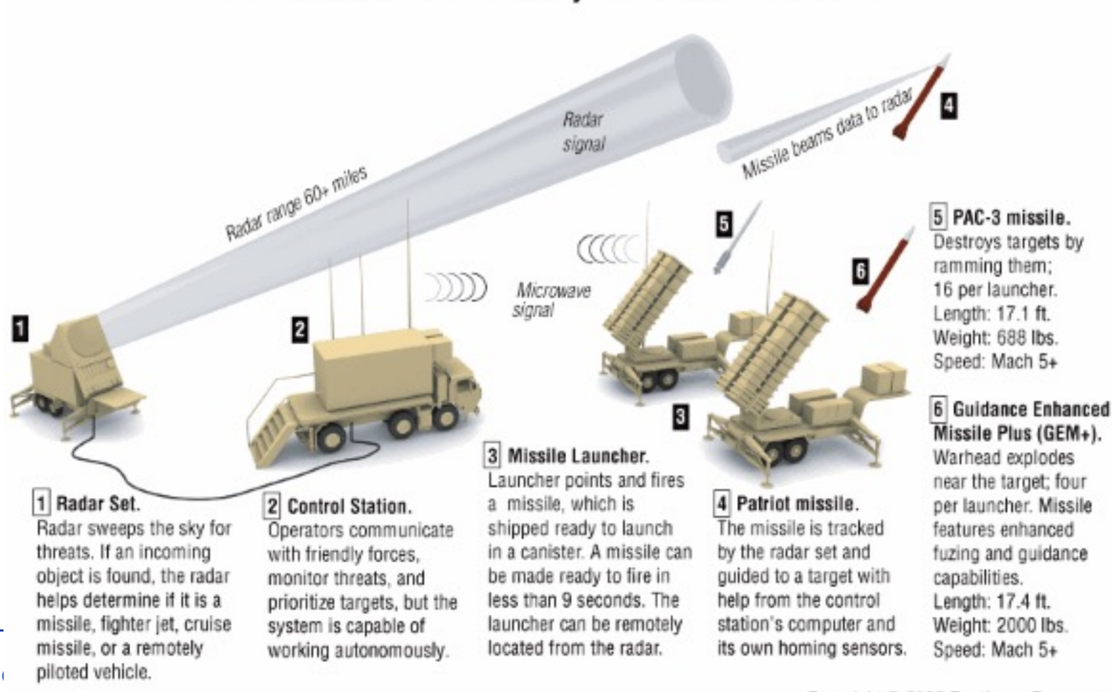
- A report from the United States General Accounting Office begins "On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing **28** Americans"


3. Track Action - Only Range Gated Portion of Beam Processed
2. Validation Action
1. Search Action- No Range Gate- Entire Beam Processed
Missile Outside Range Gate
Range Gate Area
Missile
Patriot Radar System
1
2
3

Patriot Missile Software Problem (2)

- **February 11,** 1991, the Israeli forces inform the Patriot Project Office about a defect found in the Patriot surface-to-air missile defense system. They discovered that running the system for consecutive 8 hours resulted in a 20% targeting precision loss, and estimated that after continuous operation for 20 hours the inaccuracy would grow so big that the Patriot would no longer be able to lock on, track, and intercept ballistic missiles.

Patriot Air & Missile Defense System: How Patriot Works



1 Radar Set. Radar sweeps the sky for threats. If an incoming object is found, the radar helps determine if it is a missile, fighter jet, cruise missile, or a remotely piloted vehicle.

2 Control Station. Operators communicate with friendly forces, monitor threats, and prioritize targets, but the system is capable of working autonomously.

3 Missile Launcher. Launcher points and fires a missile, which is shipped ready to launch in a canister. A missile can be made ready to fire in less than 9 seconds. The launcher can be remotely located from the radar.

4 Patriot missile. The missile is tracked by the radar set and guided to a target with help from the control station's computer and its own homing sensors.

5 PAC-3 missile. Destroys targets by ramming them; 16 per launcher. Length: 17.1 ft. Weight: 688 lbs. Speed: Mach 5+

6 Guidance Enhanced Missile Plus (GEM+). Warhead explodes near the target; four per launcher. Missile features enhanced fuzing and guidance capabilities. Length: 17.4 ft. Weight: 2000 lbs. Speed: Mach 5+

The Psychology of Testing

- The mindset which we apply during *Testing and Reviewing* is different from the one that we use during *Designing* or *Developing*.

- While building the software, we work positively towards the software with an intent to meet customer requirements. However, when we test or review the product, we lookout for defects in the product.

The Psychology of Testing (2)

- A human psychology element called *Confirmation bias* refers to thinking that makes it difficult to accept information that disagrees with currently held beliefs.
- For example, *Developers believe that their code has no errors. So, it is difficult for them to take that their code is incorrect. Testing is often looked upon as bearer of bad news by Developers as it highlights defects/failures in the system. It's difficult to see the bigger picture that these defects eventually make the software better and more usable.*

Characteristics of a Software Tester

- Software testers also need to acquire the following skills in addition to technical skills:
  - Interpersonal skills: Firstly, there should be effective communication between testers & developers about defects, failures, test results, the progress of tests and risks, etc. The way of conveying message should be very concise, complete, and humble. Additionally, they should to build friendly relations with developers so that they are comfortable to share feedback.
  - Sharp observation: Secondly, the software testers must have an "eye for detail". The testers can quickly identify or detect many critical errors if they observe sharply. Moreover, they should examine the software for the parameters such as 'look & feel' of GUI, incorrect data representation, ease of use, etc.

Characteristics of a Software Tester (2)

- Customer-oriented perspective: the software testers should adopt a customer-oriented perspective while testing the software product. They should be able to place themselves in customer shoes and test the product as a mere end-user.
- Cynical but friendly attitude: Regardless of the nature of the project, the tester must be tenacious when questioning even the minor ambiguity until it is proven. Different situations may arise during the test. For instance, the detection of a large number of errors may cause a more significant delay in the shipment of the product. It can lead to a tight situation between testers and other development teams. The tester must balance this relationship. It should not happen at the expense of errors. Testers should convince and defend the intentions of "attacking software issues but not software developers."

Characteristics of a Software Tester (3)

– Organized, flexible, and patient at work: Testers realize that they can also make mistakes. Therefore, they should be excellent organizers -they must-have checklists, facts, and figures to support their findings. Additionally, the tester should be flexible and open to new strategies. Sometimes, significant tests must be re-run that would otherwise change the fundamental functionality of the software. Therefore, the tester should have the patience to retest the software for as many new errors as may arise. Testers must be patient and stay prepared in projects where requirements change rapidly.

Characteristics of a Software Tester (4)

- Objective and neutral attitude: No one likes to hear and believe the bad news. Well, testers resemble messengers of bad news in a software project team. No matter how brilliant the testers are at their job; nobody wants to share the bad news. But, the tester always communicates the wrong part of the software, which the developers do not like. The tester must be able to deal with the situation in which he has to face the accusation of doing his job (i.e., detecting errors) too well. The tester’s work should be appreciated, and the development team should welcome the errors. That is because every potential error encountered by the tester would mean a reduction of an error that the client might have encountered.

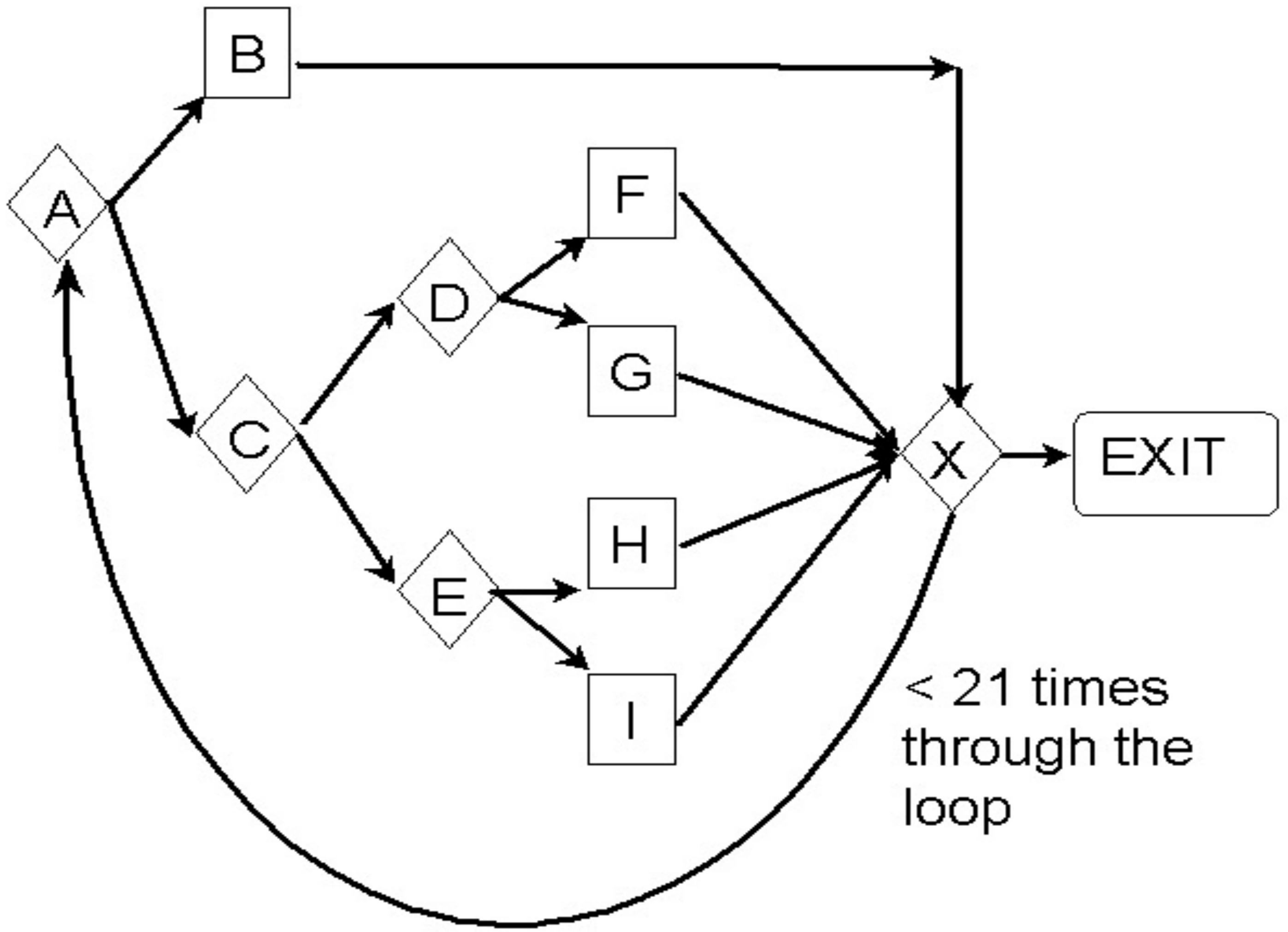
Reasons that Bugs Escape Testing

- User executed untested code.
- User executed statements in a different order than was tested.
- User entered an untested combination.
- User’s operating environment was not tested.

Why Can’t Every Bug be Found?

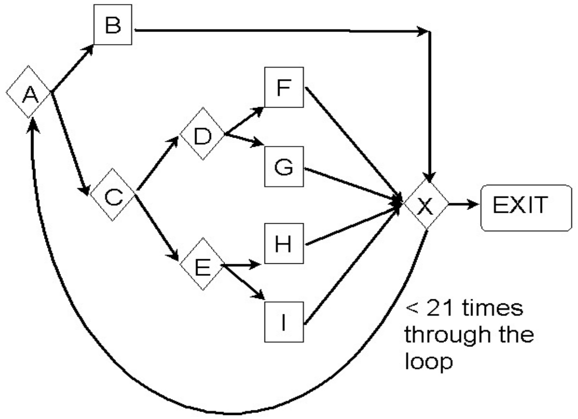
- Too many possible paths.
- Too many possible inputs.
- Too many possible user environments.

Too Many Possible Paths



Too Many Possible Paths

- There are 5 paths from A to X without passing through the loop.
- There are 5^{20} paths from A to X after passing through the loop 20 times.
- There are $5 + 5^2 + 5^3 + \ldots + 5^{20} = 100$ trillion possible paths in this program.
- If you could test a path per second it would take more than 3 million years!



Too Many Possible Inputs

- Problems take input in a variety of ways: mouse, keyboard, and other devices.
- Must test Valid and Invalid input.
- Most importantly, there are an infinite amount of sequences of inputs to be tested.

Too Many Possible User Environment

- Difficult to replicate the user's combination of hardware, peripherals, OS, and applications.
- Impossible to replicate a thousand-node network to test networking software.

Paradoxes and Main Principles

- Testing identifies the presence of defects.
  - Testing enables the identification of defects present in a piece of code, but does not show that such defects are not present.
- Exhaustive testing is impossible
  - It is impossible to undertake exhaustive testing, to test “everything”.

Paradoxes and Main Principles (2)

- Early testing
  - Test early in the development cycle or “early testing”

  This principle is based on the fact that costs, whether development costs, testing costs, defect correction cost, etc. increase throughout the project. It is economically more sensible to detect defects as early as possible; thus avoiding the design of incorrect code, the creation of test cases, which require design and maintenance of the created test cases, the identification of defects that have to be logged, fixed, and retested.

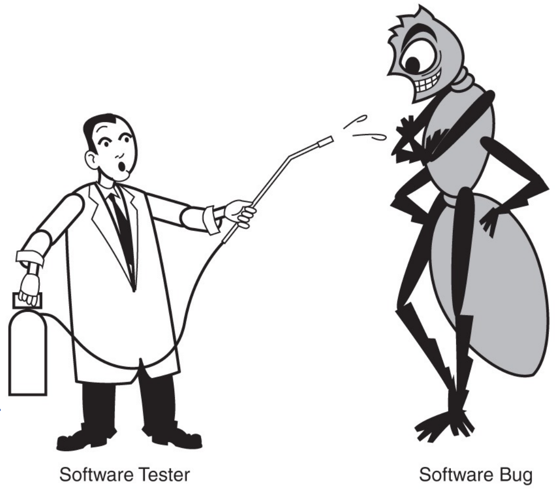
Paradoxes and Main Principles (3)

- Defect clustering
  - Even if bugs do not have a real life, it is frequent that a number of defects are concentrated according to identical criteria. These can be a piece of code (such as a complex section of an algorithm), a sub-assembly of components designed by the same person (where a similar error is repeated), or a group of components designed in the same period of time, or even a defective off-the-shelf component (COTS). This principle means also that if you find a defect, it might be efficient to look for other defects in the same piece of code.

Paradoxes and Main Principles (4)

- Pesticide paradox
  - The pesticide paradox, or lack of efficiency of tests when they are used over a period time.

  This loss of efficiency is due to the re-execution of identical test (same data, same actions) on software paths that have already been tested. If a failure has been identified and corrected during a previous execution of that test, it will not be detected anymore during the re-execution of that test at a later time.



Paradoxes and Main Principles (5)

- Testing is dependent on the context.
  - Safety-critical systems will be tested differently and with a more intense effort. However, this principle goes much further, because it also applies to the evolution of the context of your own software application. During the first tests of your software, you will design tests based on your knowledge of the software at that moment in time. During subsequent tests, or tests of the next version of the software, your knowledge of the software will have evolved, and you will design other tests, focusing, for example, on identified risks, on components where defect clustering occurred, or taking into account the pesticide paradox.

Paradoxes and Main Principles (6)

- Absence of error fallacy
  - Very frequently, the reduction of number of defects is considered to be the ultimate goal of testing. However, it is not possible to assure that a software application, even if it has no defect, suits the needs of the public. If the software does not correspond to what the users need or to their expectations, identification and correction of defects will not help the architecture or the usability of the software, nor the financial success of the software.

Test Team

Test Groups

- There is no right or wrong ways to organize test teams.
- The structure one chooses will affect productivity, quality, customer satisfaction, employee morale, and budget.
- Unit tests are developed and executed by the software developers themselves, rather than an independent unit test group.

Test Groups (2)

- It is recommended to have at least two test groups:
  - integration test group
  - system test group

- The acceptance test group is formed on a demand basis consisting of people from different backgrounds.
- The acceptance test group is dismantled after the project is completed.

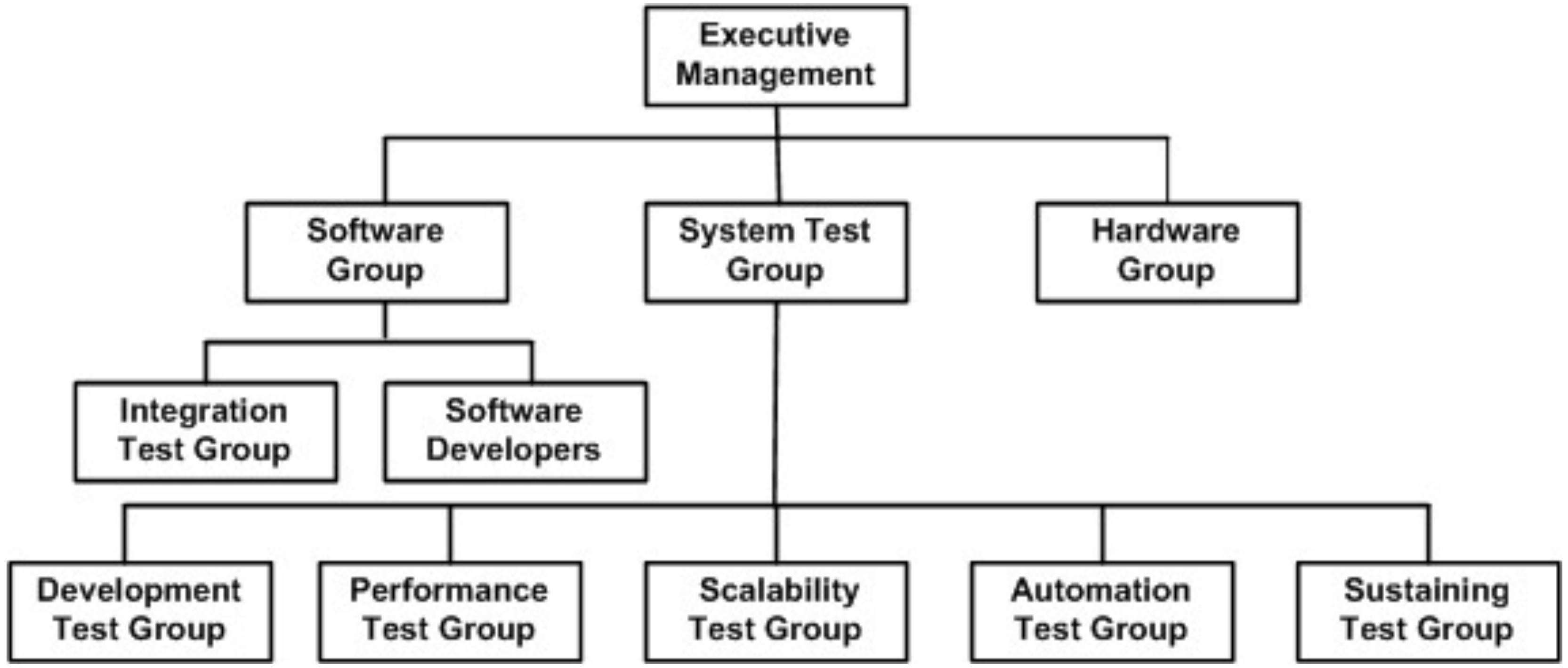
Integration Test Group

- The mandate of this group is to ensure that unit-tested modules operate correctly when they are combined.
- The leader of the integration test group reports to the software development manager.
- The software developers, who together built the modules, must be involved in performing integration testing.
- In practice, the developers themselves may integrate the system.
- The system architects are also involved in integration testing for complex systems.
- The test group may perform other duties, such as:
  - code inspection, configuration management, release management, and management of development laboratory.

System Test Group

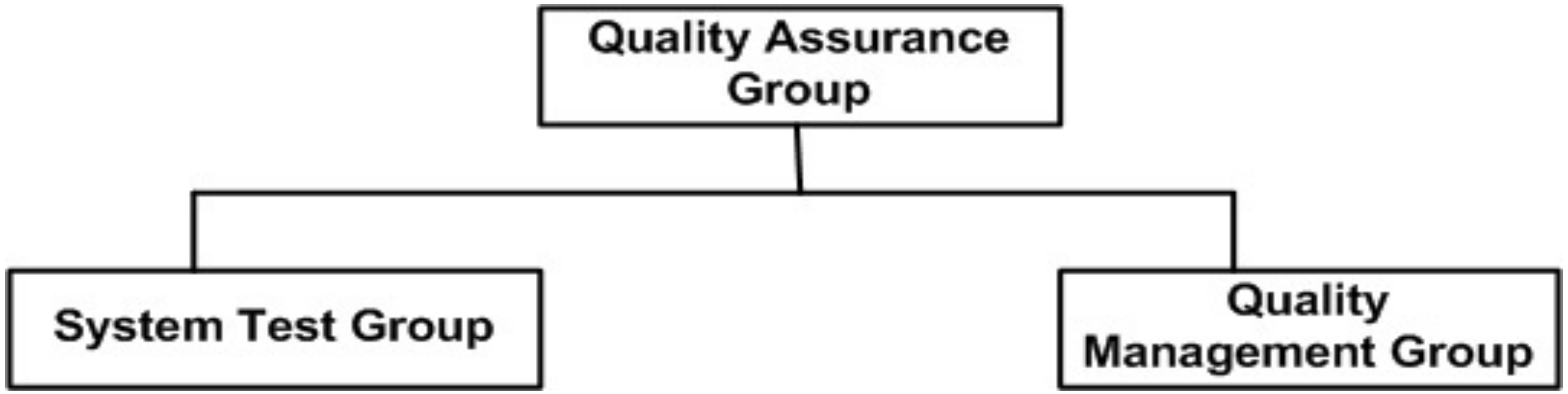
- The mandate of this group is to ensure that the system requirements have been satisfied and that the system is acceptable.
- The system test group is truly an independent group, and they usually have a separate headcount and budget.
- The manager of this group is a peer to the hardware or software development managers.
- The group executes business acceptance tests identified in the user acceptance test plan.

System Test Group (2)



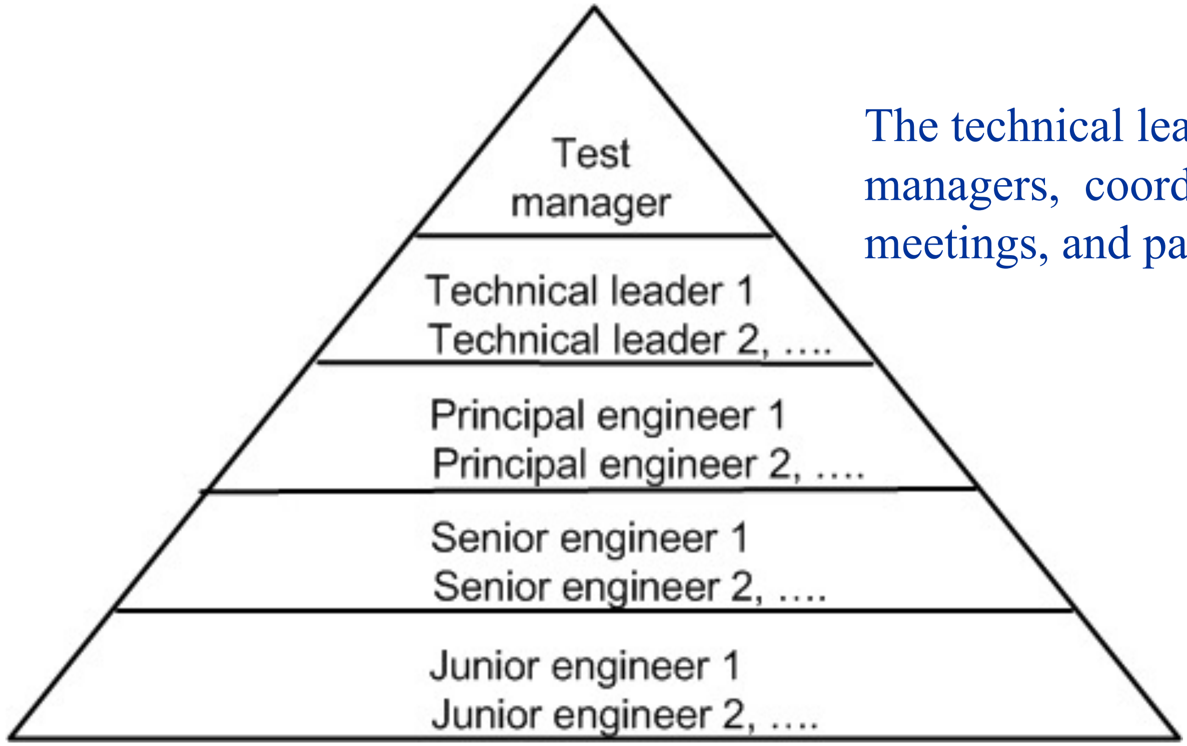
Software Quality Assurance Group

- Software quality assurance deals not only with the location of defects, but also mechanism to prevent defects.
- Software quality assurance group has a larger role in ensuring, conformance to the best development practices throughout the organization.



System Test Team Hierarchy

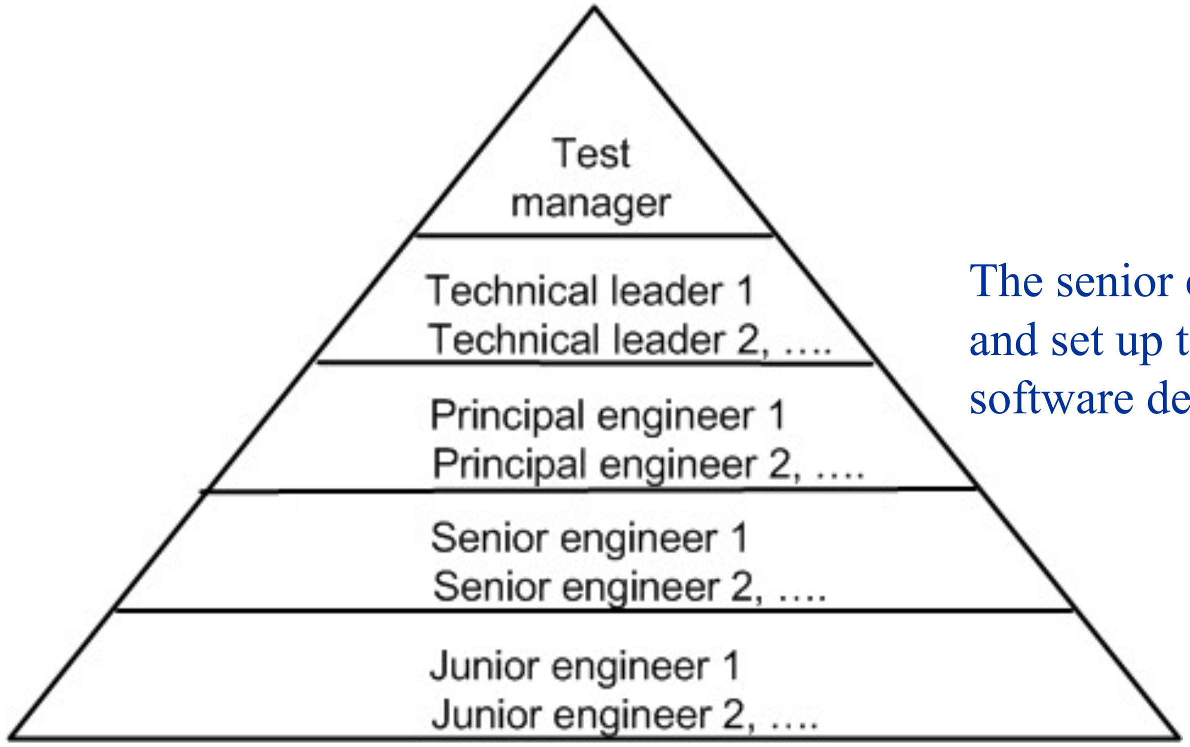
Test manager: a key person in charge of all aspects of the assigned testing task.



The technical leader provides the strategic direction for software testing, assists test managers, coordinates test plan review meetings, attends entry/exit criteria review meetings, and participates in cross-functional review meeting

The principal engineers are test specialists responsible for test planning, test automation, test environment planning, test tool development, procurement of test equipment, performance modeling, reliability engineering, and business acceptance testing. In addition, the principal engineers execute test cases and mentor junior engineers within the group.

System Test Team Hierarchy (2)



The senior engineers design, develop, and execute test cases. They design and set up test laboratories and environments. Senior engineers assist software developers in reproducing known defects

New, inexperienced test engineers are put at a junior level. They gain experience by assisting the senior and principal engineers in charge of test execution, setting up of test beds, and developing scripts for test automation

Effective Staffing of Test Engineers

- A successful test team is made up of members whose strengths are complementary
- It is advisable to have people on the test team with diverse background and experience, such as:
  - developers
  - integration testers
  - information technology administrators
  - technical support personnel
  - technical writers
  - quality management personnel
  - experienced test engineers
  - recent graduates

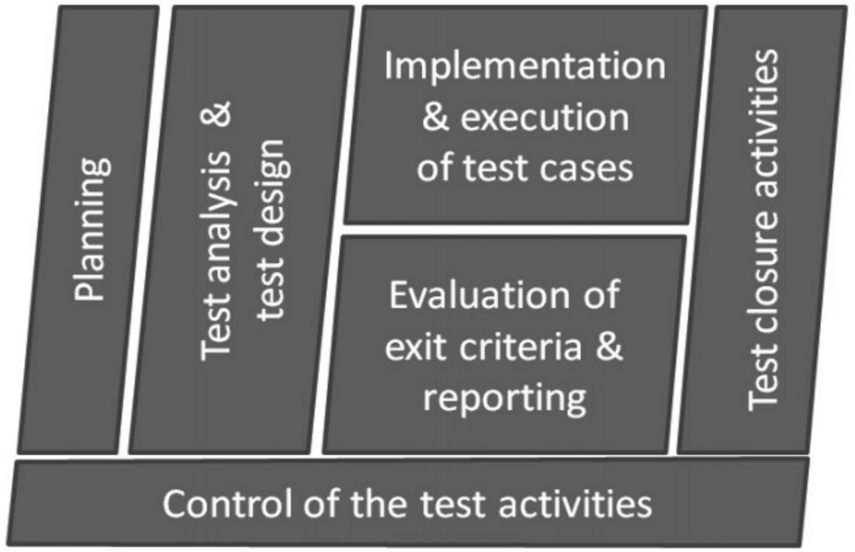
Effective Staffing of Test Engineers (2)

- Test engineers are expected to have the following skills
  - Have credibility with software developers
  - Understand developers' terminologies
  - Know when the code is ready for testing
  - Be able to evaluate the impact of a problem on the customers
  - Assist software developers in expediting defect resolution
  - Reduce false-positive and false-negative results in testing
  - Develop expertise in test automation
  - Mentor the junior test engineers

Testing Progress

Fundamental Test Process

- Testing is not limited to the execution of software with the aim of finding failures: it is also necessary to plan, define goals, identify test conditions, create test data, start and exit criteria, test environments, and of course, control all these activities


Planning
Test analysis & test design
Implementation & execution of test cases
Evaluation of exit criteria & reporting
Test closure activities
Control of the test activities

Fundamental Test Process (2)

- The fundamental test process includes:
  - Test planning and control;
  - Analysis and design of tests;
  - Implement and execution of tests;
  - Evaluation of the exit criteria and production of test report;
  - Test closure activities.

Planning

- Before any human activity, it is necessary to organize and plan the activities to be executed. This is also applicable for software and system testing.

- Test planning activities include organization of the tasks, and coordination with the other stakeholders, such as development teams, support teams, user representatives, management, customer, etc.

Control

- Control activities are executed throughout the test campaign, They are often grouped with planning activities because they ensure that what has been planned is correctly implemented.

- Control identifies deviations from planned operations, or variants with regards to planned objectives, and proposes actions to reach these objectives. This implies the ability to measure the progress of activities in terms of the resources used as well as in terms of objectives reached.

Test Analysis and Design

- The analysis and design phase is where global test objectives, as defined by organization at the test plan level, are used to create the test conditions for the software.
- Analysis of the basis of the test include
  - Contractual documents, such as the contract, statement of work, and any amendments or technical attachments, etc;
  - Software specification, high-level design, detailed architecture of the components, database organization or file system organization;
  - User documentation, maintenance or installation manuals, etc;
  - The risks identified or associated with the use of the development of the software;
  - Applicable standard, whether they be company specific, profession specific or mentioned in the contractual documents.

Test Analysis and Design (2)

- Test design
- Test objectives correspond to the reasons or goal that we have, during test design and execution. These will guide our actions and will lead to the detailed design of the test cases.

- Test design consists of applying the test objectives under obvious test conditions, and then applying them to test cases.

Test Design

- Test design phase comprise:
  - Identification and prioritization of the test conditions based on an analysis of the test objects, of the structure of the software and system;
  - Design and prioritization of high-level test cases;
  - Identification of the test environment and test data required for test execution;
  - Provision for control and tracking information that will enable evaluation of test progress.

Test Implementation

- Test implementation is the conversion of test conditions towards test cases and test procedures, with specific test data and precise expected results. Detailed information on test environment and test data, as well as on the sequence of the test cases are necessary to anticipate test execution.

Test Execution

- Test execution in the test environment enables the identification of differences between the expected and the actual results and includes tasks linked to the execution of test cases, test procedures, or test suites. This includes:
  - Ensuring that the test environment is ready for use, including the availability of test data;
  - Executing test cases, test procedures, and test suites, either manually or automatically, according to the planned sequence and priority;
  - Recording test execution results and identifying the version of the component tested, of the test tools, and test environment;

Test Execution (2)

- Comparing expected results with actual results;
- Identifying and analyzing any discrepancy between expected and actual results, and clearly define cause of the discrepancy. Recording these differences as incidents or defects, by providing the highest level of detail possible to facilitate defect correction in the future;
- Providing tracking and control information to allow efficient management of test activities.

Analysis of Exit Criteria

- Analysis of the exit criteria evaluates the test object with regards to the test objectives and criteria defined during the planning phase.

- This evaluation include:
  - Analysis of the test execution logs and reports;
  - Comparison of the objectives reached versus objectives identified during the planning phase, and evaluation of the need to test more thoroughly or to modify the exit criteria;

Reporting

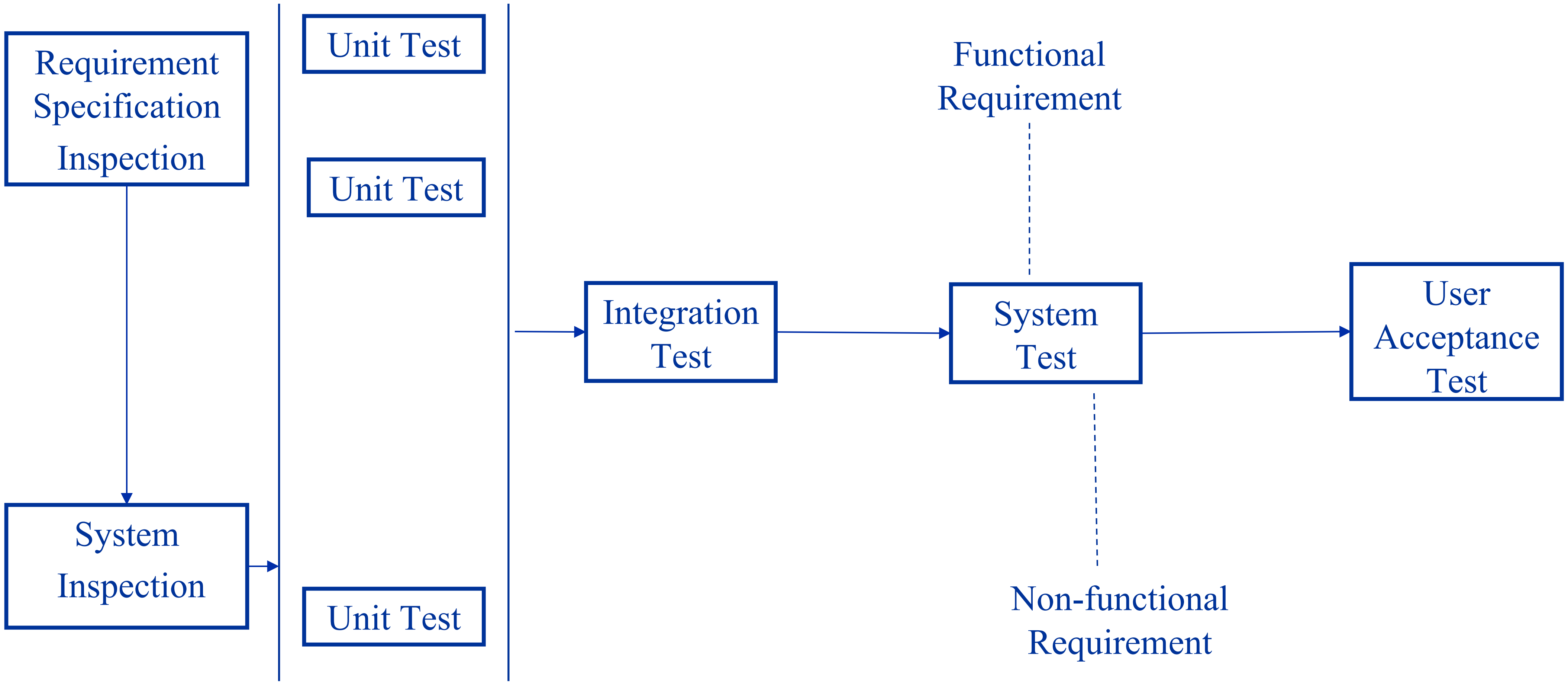
- Test activity results interest a large number of stakeholders:
  - Testers, to evaluate their own progress and efficiency;
  - Developers, to evaluate the quality of their code, and the remaining workload, whether it is on the basis of remaining defects to be fixed, or components to deliver;
  - Quality assurance managers, to determine the required process improvement activities, whether in the requirement elicitation phase or reviews, or during design or test phases;
  - Customer and end users, or marketing, to advise them when the software or system will be ready and released to the market;
  - Upper management, to evaluate the anticipated remaining expenses and evaluate the efficiency and effectiveness of the activities to date.

Test Closure Activities

- Once the software or system is considered ready to be delivered, or the test project is considered complete, it is necessary to close the test activities. This consists of:
  - Ensuring that the planned components have been delivered;
  - Determining the actions that must be taken to rectify unfixed incidents or remaining defects. This can be closure without any action, raising change requests on a future version, delivery of data to the support team to enable them to anticipate user questions;
  - Document the acceptance of the software or system;
  - Archive test components and test data, drivers and stubs, test environment parameters and infrastructure for future usage;
  - Identify possible lessons learned or return on experience, so as to document them and improve future projects and deliveries, and

Testing Progress

Test Phase


Requirement Specification Inspection
System Inspection
Unit Test
Unit Test
Unit Test
Integration Test
System Test
User Acceptance Test
Functional Requirement
Non-functional Requirement

Types of Tests

- Test object: Unit test, integration test, system test, user acceptance test.
- Test techniques: Black-box (specification-based), white-box (structure-based)
- Execution: Execution-based testing, Nonexecution-based testing;

Component level test/Unit test/Class test

- *Test object*:
  - components, program modules, functions, programs,
- *Objective*:
  - detect failures in the components,
- *Entry criteria*:
  - the component is available, compiled, and executable in the test environment;
  - the specifications are available and stable.
- *Exit criteria*:
  - the required coverage level has been reached;
  - defects found have been corrected;
  - the corrections have been verified;
  - regression tests have been executed on the last version and do not identify any regression;
  - traceability from requirements to test execution is maintained

Integration Level Testing

- *Test object*:
  - components, infrastructure, interfaces, database systems, and file systems.
- *Objective*:
  - detect failures in the interfaces and exchanges between components.
- *Entry criteria*:
  - at least two components that must exchange data are available, and have passed component test successfully.
- *Exit criteria*:
  - all components have been integrated and all message types (sent or received) have been exchanged without any defect for each existing interface;
  - statistics (such as *defect density*) are available;
  - the *defects* requiring correction have been corrected and checked as correctly fixed;
  - the impact of not-to-fix defects have been evaluated as not important.

Types of integration

- Integration of software components
  - typically executed during component integration tests
- Integration of software components with hardware components
- Integration of sub-systems
  - typically executed after the system tests for each of these sub-systems and before the systems tests of the higher-level system.

System Test

- *Test object*:
  - the complete software or system, its documentation, the software configuration and all the components that are linked to it
- *Objective*:
  - detect failures in the software, to ensure that it corresponds to the requirements and specifications, and that it can be accepted by the users.
- *Entry criteria*:
  - all components have been correctly integrated, all components are available.
- *Exit criteria*:
  - the level of coverage has been reached;
  - must-fix defects have been corrected and their fixes have been verified;
  - regression tests have been executed on the last version of the software and do not show any regression;
  - bi-directional traceability from requirements to test execution is maintained;
  - statistical data are available, the number of non-important defects is not large;
  - the summary test report has been written and approved.

Design of Test Cases

Test cases and Test suite

- Test a software using a set of carefully designed test cases:
  - The set of all test cases is called the test suite

Test cases and Test suite

- A test case is a triplet [I,S,O]:
  - I is the data to be input to the system,
  - S is the state of the system at which the data is input,
  - O is the expected output from the system.

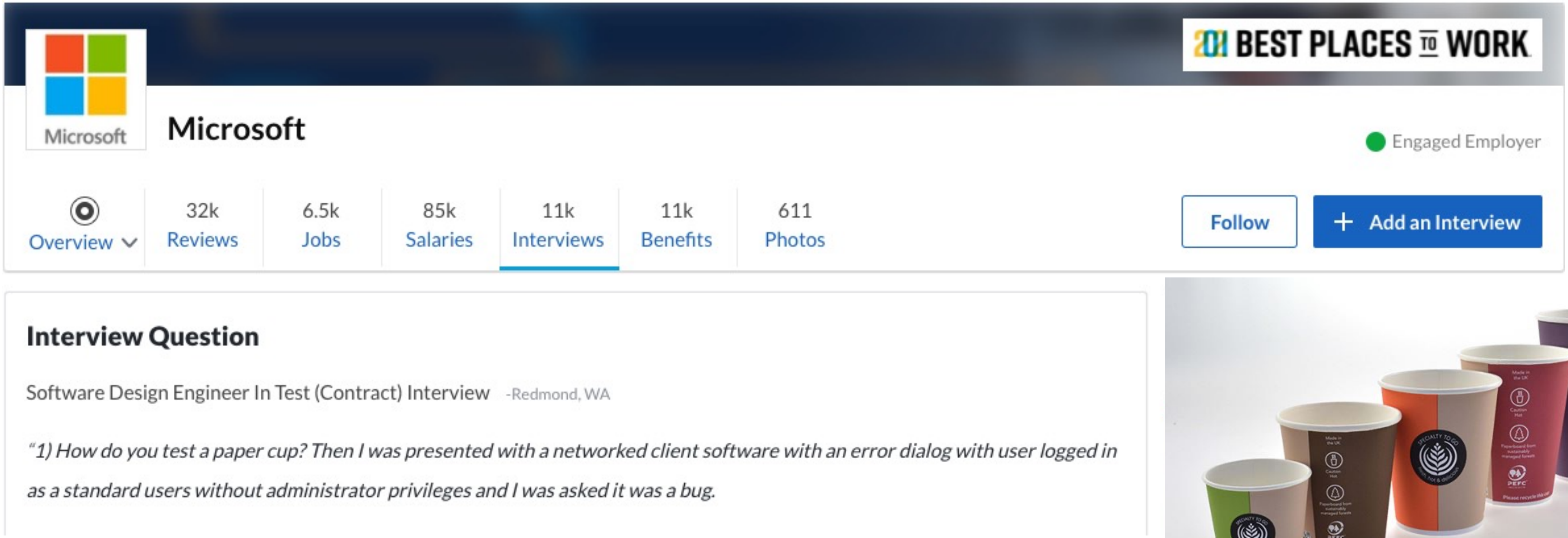
Design of Test Cases

- **What aspects does a test case contain?**
- **1) Objective:** Here the tester mentions what he plans to achieve with that particular test case.
- **2) Steps to Follow:** Here the tester mentions the steps that need to be followed to achieve the objective.
- **3) Expected Output:** Here the tester mentions the output which is expected as per the requirements provided.
- **4) Actual Output:** Here the tester mentions the actual output achieved by following the steps.
- **5) Pass/ Fail:** If the tester fails to achieve the 'Expected Output' by following the steps then he will mention 'Fail' against that particular test case. Similarly, if the tester is able to achieve the 'Expected Output' then he will mention 'Pass' against the test case.

Design of Test Cases (2)

- Test Cases:
  - Representative;
  - Explore the weakness/defects in system design and functionalities;
  - Positive Testing (i.e., valid inputs) and Negative Testing (i.e., invalid inputs);
  - Environments and Scenarios;

Case Study: Paper Cup



Interview Question

Software Design Engineer In Test (Contract) Interview -Redmond, WA

"1) How do you test a paper cup? Then I was presented with a networked client software with an error dialog with user logged in as a standard users without administrator privileges and I was asked it was a bug.