TESTING FUNDAMENTALS

TABLE OF CONTENTS

MODULE – OVERVIEW OF TESTING

TERMINOLOGY/DEFINITIONS

Test process	It is a process of controlling and management of test sets, test results and test reports.
Installation	It is a process of placing the tested program into production.
Maintenance	As the system is used, it is modified either to correct error or to augment the original system. After each modification, at any level, the system should be retested, re-verified and updated subsequently.
Positive testing	Evaluating an attribute of a system and determining that it meets its stated requirements.
Negative testing	It is the process of executing a program or system with the intent of finding defects.
Tester psychology	The emotional and behavioral characteristics of tester.
Test lifecycle	It describes the different procedures that are to be approached for a successful testing process in a sequential manner so that an organization is able to achieve high quality test process.
Test design	It involves in designing of test procedure, test data, test scenarios and test cases.
Test execution	It is a process of executing all or a selected number of test cases and observing the result.
Test reporting	It is prepared when testing is complete or to summarize the progress of testing effort. It gives details about the tests done and their status/results.
Test management	An activity that helps in keeping track of the progress of the testing, quality of product and need of attention to various aspects.
Test plan	It describes what specific tasks must be accomplished and who is responsible for each task, what procedures are to be used, how much time and effort is needed and what resources are essential to carry out the activities of testing.
Test strategy	It is formulation of the approach to test a product, defining the levels and types of testing required, the techniques to be applied, the extent and kind of test automation required.
Traceability matrix	A traceability matrix is a table that correlates requirements or design documents to test documents. It is used to change tests when the source documents are changed, or to verify that the test results are correct.
Test scenario	It is a thread of operational use.
Test case	It is a combination of inputs with an expected output.
Test script	The test script is a procedure or a program that replicates the user actions. Test Case will be a baseline to create test scripts using a tool or a programming/scripting language.
Test suite	A collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases.
Test data	Test data is a set of values or data used to test the same functionality of a particular feature.
Test harness	The software, tools, samples of data input and output, and configurations are all collectively referred to as a test harness.

DEFINITION OF TESTING

Testing is a set of activities that can be planned in advance and conducted systematically for which a template or a set of steps needs to be defined. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

OBJECTIVE OF TESTING

The chief objective of testing is to find defect/fault in the system as early as possible which can be done by developing good test cases that has a very high probability of finding an undiscovered error.

PSYCHOLOGY OF A TESTER

Testers hunt for errors

The tester's job is to find every conceivable fault or weakness in the software work product, so that the probability of detecting an as yet undiscovered error during the testing is very high. The focus on showing the presence of errors is the basic attitude of

the tester. We feel good on the day when we find defects, and we are thrilled when the defect-finding goal exceeds for the day or week.

Testers are creatively destructive
Testing is a positive and creative effort of destruction. It takes imagination, persistence and a strong sense of mission to systematically locate the weakness in a complex structure and to demonstrate its failure.
When a developer develops a work material, he starts with the viewpoint that it works, as he is well aware of its boundaries and problems. The minor problems may become major issues when anyone confronts it in a later stage. So a tester comes into the picture where he needs to attack the material with the attitude of: "I'm here to destroy your material and I'm going to find as many defects as possible."

Testers pursue errors, not people
Errors are in the work product, not with the individual who developed it. With the "test to destroy attitude", testers are not attacking a developer or team of developers in an organization.
Similarly, a developer needs to understand that testers are not against them in finding errors in their products. So it becomes extremely important that the developer and the tester work in mutual understanding and balance. Working as team in a well-organized manner helps a great deal in developing the final product successfully.

Testers add value
Testers add value to software by discovering the errors and reporting them as early as possible to:
- Save developers from building products based on error-ridden sources
- Ensure management gets the bottom line on the quality and finance they are looking for
- Ensure the marketing people can deliver what the customer expects from the product

Good testers are:
Explorers: *Are not afraid to venture into unknown situations*
Troubleshooters: *Good at figuring out why something does not work*
Relentless: *Keep looking for recreating the elusive bug*
Creative: *Come up with off-the-wall approaches to find bugs*
(Mellowed) Perfectionists: *Strive for perfection, know how far to go*
Exercise good judgment: *What to test, how long it will take, is it really a bug*
Tactful and diplomatic: *As bearers of bad news, do it tactfully and professionally*
Persuasive: *Why these bug(s) need to be fixed and follow-up to make it happen*

TEST LIFECYCLE

Test life cycle describes the different procedures that are to be approached for a successful testing process in an incremental manner so that an organization is able to achieve high quality test process. In the testing domain, test plans support achieving testing goals for a project. It helps to select tools and techniques to achieve the goal, and to estimate time and resources needed for testing tasks so that testing is effective, on time, within budget, and consistent with project goals.

Early views saw testing as a phase that occurred after software development, or "something that programmers did not get their bugs out of their programs." The more modern view sees testing as a process to be performed in parallel with the software development or maintenance effort incorporating the activities of:

Requirement Analysis - Setting test objectives and requirements

Strategy and Planning - determining risks and selecting strategies

Design - specifying test procedure, test scenarios and test cases to be developed

Execution - running and rerunning the tests

Reporting and management - tracking and updating the progress of software tests

Analysis – analyzing the test results and taking appropriate corrective actions

The figure below shows a comparison between the early view and the modern view of how testing was carried out.

This lifecycle perspective of testing represents a major change from just a few years ago, when many equated testing executing tests. The contribution of planning, analyzing, and designing tests was under-recognized (and still is by many people), and testing was not seen as really starting until tests started running. Now we understand the evaluation power of test planning and analysis. These activities can be more powerful than test execution in defect prevention and timely detection. We also understand that an accurate interpretation of the situation when "all tests are running successfully" requires a clear understanding of the test design.

The lifecycle model for testing that has emerged borrows heavily from the methodology we've grown accustomed to for software. Considering that a test set is made up of data and procedures (which are often implemented as executable test programs), it should not come as a surprise that what it takes to build good software is also what it takes to build good test ware.

The testing life cycle involves the continuous testing of the system during the developmental process. At predetermined points, the results of the development process are inspected to determine the correctness of the implementation. These inspections identify defects at the earliest possible point.

Testing life cycle cannot occur until a formalized system development process has been incorporated. It is dependent on the completion of predetermined deliverables at specified points in the development life cycle of the software.

The testing life cycle concept can be best accomplished by the formation of a test team, comprising members of the project who may be both implementing and testing the system. However, when members of the team are testing the system, they must use a formal testing methodology to clearly distinguish the implementation mode from testing mode, and they must also follow a structured methodology when approaching testing the same as when approaching system development. Without structured testing methodology, the testing process will be ineffective.

TEST REQUIREMENTS ANALYSIS

This phase of the test lifecycle deals with development of vision and goals required for testing as well as understanding, prioritizing and implementing the requirements. Here the requirements of the software system are viewed from a testing perspective. The test requirements should follow all the basic rules of the software requirements. They need to be specific, measurable, achievable, realistic and timely. The requirements must also support the goals and vision of the organization.

TEST STRATEGY AND PLANNING

TEST STRATEGY

Objectives:

A test strategy is basically a policy an organization that can be defined as a high level statement of principle or course of action that is used to govern a set of activities in that organization. It provides a vision and framework for decision-making. It is important to have the policy formally adopted by the organization, documented and available for all interested parties. A team or task force consisting of upper management, executive personnel and technical staff to carry out the testing activities efficiently, should formulate a policy statement.

In the case of testing, the testing strategy is used to guide the course of testing activities and test process evolution. Testing policy statements reflects, integrate and support achievements of testing goals, which often in turn target increasing software quality and improving customer satisfaction. The test strategies also provide high-level guidance as to how testing is to be done in the organization, how its effectiveness will be evaluated, who will be responsible and what choices of resources are possible. They should be explicit enough to guide decisions like how to test, what to test, how much to test, and who will test.

The testing strategies should have strong support and continual commitment from the management. After strategy has been developed, approved and distributed, a subset of the task force should be appointed permanently to oversee strategy implementation and change. This is important for several reasons as it helps testing to be a planned activity and can therefore be managed.

TEST PLANNING

Test planning is an essential practice for any organization that is repeatable and manageable. Pursuing the maturity goals is not a necessary pre-condition for initiating a test planning process, however, a test process improvement effort thus provide a good framework for adapting this essential practice. Test planning should begin early in the software lifecycle in order to achieve the desired goals of the company within the budget and available time.

The model such as V-model helps to support test-planning activities that begin in the requirement phase and continue on to successive software development phases. In order to meet a set of goals, a plan describes what specific tasks must be accomplished and who is responsible for each task, what tools, procedures and techniques must be used, how much time and effort is needed and what resources are essential to carry out the activities of software development.

A test plan document is a document that provides a framework or approach for achieving a set of goals. It may also contain milestones, which are nothing but tangible events that are expected to occur at a certain time, in the projects life cycle. Managers use milestones to determine the status of the projects. Tracking the actual occurrence of the milestone events allows the manager to determine if the project is progressing as planned.

TEST DESIGN

Design phase of test lifecycle involves in designing the test procedure, test data, test scenarios and test cases. Organization of verification effort and test management activities should be closely integrated with preliminary design. The general testing strategy including test methods and evaluation criteria-is formulated and a test plan is produced. If the project size or criticality warrants, an independent test team is organized. In addition a test schedule with observable milestones is constructed. At the same time, the framework for quality assurance and test documentation should be established.

During detailed design, validation support tools should be acquired or developed and the test procedures themselves would be produced. Test data to exercise the functions introduced during the design process, as well as test cases based upon the structures of the system, should be generated. Thus, as the software development proceeds, a more effective set of test cases is built.

In addition to test organization and the generation of test cases, the design itself should be analyzed and examined for errors. Simulation can be used to verify properties of the system structures and subsystem interaction, design walkthroughs should be used by the developers to verify the flow and logical structure of the system, while design inspection should be performed by the test team. Missing causes, faulty logic, module interface mismatches, data structure inconsistencies, erroneous I/O assumptions, and user interface inadequacies are items of concern. The detailed design must prove to be internally coherent, complete, and consistent with the preliminary design and requirements

TEST EXECUTION

Test execution is the process of executing all or a selected number of test cases and observing the results. Although preparation and planning for test execution occur throughout the software development lifecycle, the execution itself typically occurs at or near the end of the software development lifecycle. Test execution is the most visible testing activity. It also typically occurs at the end of development life cycle.

Who executes the test is depended upon the level of testing. Developers, testers, end users or some combination of all could execute system testing. Developers and/or the testers usually execute integration tests. During unit tests it is normal for developers to execute the test. Ideally, end users should execute acceptance test, although developers and/or testers may also be involved.

Choosing which test cases to execute first is a strategy decision that depends on the quality of the software, resources available, existing test documentation and the results of the risk analysis. As you run the tests, you learn more about the system and are in a better position to write additional test cases. The result of each test case must be recorded. If the testing is automated the tool will record both input and the results if the tests are manual then the results can be recorded right on the test case document. In some instances, it may be adequate to merely indicate whether the test case passed or failed. The by-products of test execution are test incident reports, test logs, testing status and test results.

TEST REPORTING AND MANAGEMENT

The test tracking and management drive the engineering process. In order to do so the project must be first planned using underlying process components as the planning framework.

As the project progresses it is monitored/tracked and controlled with respect to plan. Monitoring and controlling are engineering management activities that help in keeping a track of the progress of the product and where it needs more attention.

They help in validating the planned requirements, types of tests, design and execution of types of test scenarios and test cases. Sometimes test process tracking requires a set of tools, forms, techniques and measures. It helps in showing the deviation from the test plan and also assists to put the project back into track. Management is an important activity, which is done to ensure that the goals will be achieved occurring to the plan.

The supplementary tasks that provide additional support for tracking and management are
- Developing standards of performance
- Plan each project
- Establishing a monitoring and reporting system
- Measure and analyze results
- Initiate corrective actions for projects that are off track
- Document the controlling and monitoring mechanisms
- Utilize a configuration management system to manage versions, releases, etc.

REPORTING TEST RESULTS

The test report can be prepared when testing is complete or to summarize the progress of testing efforts. It gives us a detail about the health of the product. Report should contain the following items.
- Activities and accomplishments during reporting period
- Problems encountered since the last meeting period
- Solved problems
- Outstanding problems
- Current project testing state vs. plan
- Expenses vs. budget
- Plans for the next time period

Report can help managers, testers, developers and SQA staff to evaluate the effectiveness of testing effort.
The reports can be test cycle report, test progress report, periodic product quality report and final test summary report.

ANALYZING THE RESULTS

It is very difficult to manage the testing activities without any healthy analysis of the testing results. Organizations spend billions of dollars on software development and testing; yet fail to inadequately analyze the tested software when completed. Thus, software is placed into production with embedded defects. Quality Assurance Institute surveys conducted over the past several years show that most production software consists of three to six defects per thousand lines of code. The defects unnoticed due to lack of analysis results in failure of the software product.
The analysis phase of the test life cycle helps in taking the corrective actions, which will help the management to identify and track the problems and improve the testing process by taking appropriate measures. This also helps in drawing conclusions as to where the product needs improvement.

References:

For further reading, refer the books listed below:

KIT, EDWARD. , Software Testing In the Real World, Pearson Education, 2003.
PERRY, WILLIAM E., Effective Methods for Software Testing, Wiley 2000.
BURNSTEIN, ILENE, Practical Software Testing, Springer,2002.

MODULE EXERCISES

1. What are the different phases of test life cycle?
2. Who are all involved in testing a software application?
3. Map the role against the phases of test life cycle.
4. Mention 5 test artifacts.

POINTS TO PONDER

1. "Testing can never show the absence of defects". Why?

2. Testing has to be performed right from the beginning of software development cycle and not just after code completion.

3. Testing is not done only by testers. Comment.

4. It is necessary for a good test engineer to be diplomatic. Why?

5. Testers need to think negatively to be effective in their work. Why?

MODULE – METHODS OF DETECTION

TERMINOLOGY/DEFINITIONS

V- Model	It is a software development and testing model which helps to highlight the need to plan and prepare for testing early in the development process.
Verification	It is a quality assurance process.
Validation	It is a quality control process.
Checklist	It is a tool for verification.
Requirement review	It is the study and discussion of the computer system requirements to ensure they meet stated user needs.
Design review	It is the study and discussions of the computer system design to ensure it will support the system requirement.
Code walkthrough	This contributes to an informal analysis of the program source code to find defects and verify coding techniques.
Code inspection	It is a formal analysis of the program source code to find defects as defined by meeting system design specification.
Coverage analyzer	It is a tool that helps in coverage analysis of the software work product. Some of them are Jcover, Purecoverage, McCabe tools set.
Memory leak detector	It is a tool which detects whether any memory leak is affecting the software program. Memory leak happens when resource allocated dynamically during testing are not freed after use. The memory leak tools are BoundsChecker, Purify, Insure++.
Software review	It serves to uncover errors and purify the software engineering activities like analysis, design and coding.
FTR - Formal technical review	The FTR is to uncover errors in function, logic; to verify software requirement; to ensure software predefined standards.
Software inspection	An inspection is a review practice found in software projects. Each inspector prepares for the meeting by reading the work product and noting each defect. The goal of the inspection is to identify defects.
Author	The person who created the work product being inspected.
Moderator	He/She is the leader of the inspection. The moderator plans the inspection and co-ordinates it.
Reader	The person reading through the documents, one item at a time. The other inspects & point out defects.
Inspector	The person who examines the work product to identify possible defects.
Inspection meeting	During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.
Rework	The author makes changes to the work product according to the action plans from the inspection meeting.
Follow-up	The changes by the author are checked to make sure everything is correct.
Audit	It is an evaluation of a person, organization, system, process, enterprise, project or product. The similar concepts are also applicable in project management, quality management and energy conservation.
Quality audits	It is performed to verify conformance to standards through review of objective evidence. A system of quality audits may verify the effectiveness of a quality management system. This is part of certifications such as ISO 9001.
Regular Health Check Audits	The aim of a regular health check audit is to understand the current state of a project in order to increase project success.
Regulatory Audits	The aim of a regulatory audit is to verify that a project is compliant with regulations and standards. The regulatory audit must be accurate, objective and independent while providing oversight and assurance to the organization.

V-MODEL

The V-model is a software development and testing model, which helps to highlight the need to plan and prepare for testing early in the development process. In V-model, each development phase is linked to a corresponding testing phase. The left hand descending arm of the " V " represents the traditional water- fall development phase, whereas the ascending right-hand arm of the " V " shows the corresponding testing phase.

V-model is a model that illustrates how testing activities can be integrated into each phase of the software development life cycle. It helps us to understand that the testing activities need to begin as early as possible in the software life cycle.

The life cycle paradigm demands a systematic, sequential approach to software development that begins at the customer requirements and progresses through software requirements, design, coding, testing and maintenance.

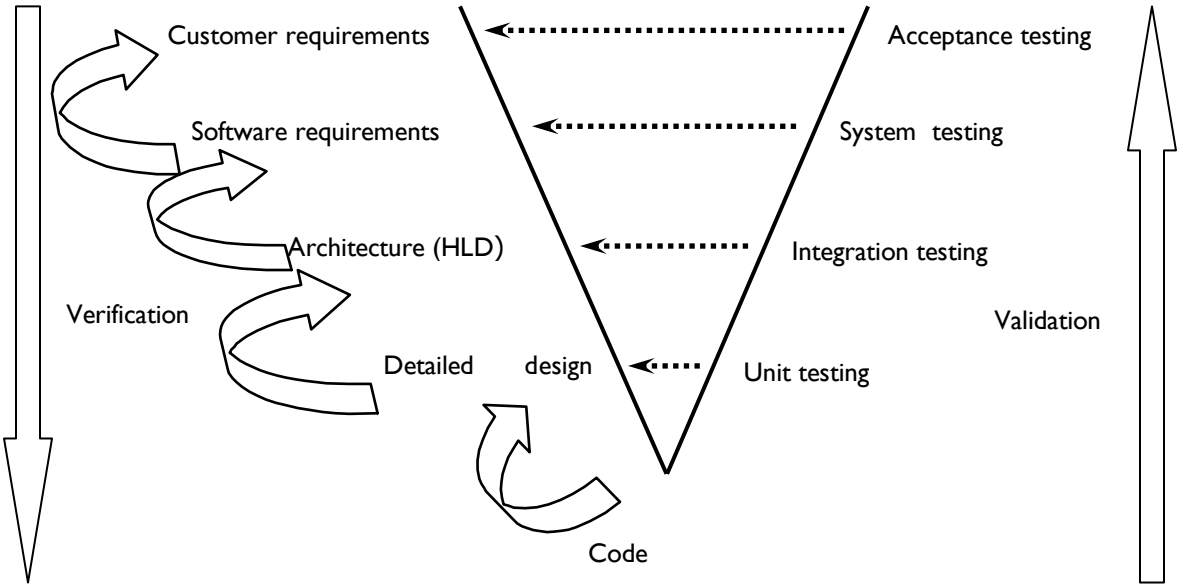


Fig 1

THE VARIOUS PHASES IN THE V-MODEL

Customer Requirement:

In order to develop initial version of the system, we can use interactions with the users/clients about the requirements documents and acceptance tests are conducted based on functional and quality requirements along with specification of system behavior. To ensure that all requirements are testable, we can hold discussion with requirements staff. Testing schedules and costs can be estimated and subsequently reduced, if we appropriately specify testing requirements and testing goals. The customer requirements are then subjected to acceptance testing and software requirements to system testing.

Software Requirement:

The requirement gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer must understand the information domain for the software as well as the required function, performance, and interfacing. Requirements for both the system and the software are documented and reviewed with the customer.

Design:

Software design is actually a multi-step process that focuses on four distinct attributes of the software program:

- Data structure
- Software Architecture
- Procedural detail
- Interface characterization

The representation of the documented requirements and specifications are developed during this phase. The design document can be used to develop integration testing. Once design is completed the system architecture or the high level design can be tested for potential interfacing issues. Like requirements, the design is documented and becomes a part of the software configuration.

Coding:

The design must be translated into a machine-readable form in coding. The coding step performs this task. If design is performed in a detailed manner, then coding can be accomplished mechanistically. Using the procedural design documentation or structural coding details of the software program, we can carry out unit test design tasks. After the source code has been developed, reviewed and verified, unit test case design begins.

Testing:

Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals, that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

During this phase the planned testing activities are executed. Different levels and techniques of testing are used in this phase.

VERIFICATION AND VALIDATION TECHNIQUES

The software development lifecycle includes different verification and validation techniques that help in assuring and controlling the quality as well as reliability of the software product throughout the process. The forthcoming sections deal with different levels and phases in the life cycle process where verification and validation becomes inevitable.

LIFE CYCLE VERIFICATION ACTIVITIES

Verification involves thorough analysis of phases as defined below.
The following activities should be performed at each phase of the life cycle:

- Analyze the structures produced at this phase for internal testability and adequacy.
- Generate test sets based on the structure at this phase.
- Determine that the structures are consistent with structures produced during the previous phases.
- Refine or redefine test sets generated earlier.

Requirements:

The adequacy of the requirements must be thoroughly analyzed and initial test cases generated with the expected results. Developing scenarios of the expected system use may help to determine the test data and anticipated results and helps in clarification of the requirement. Vague or un-testable requirements or late discovery of inadequate requirements can be very costly.

Design:

The general testing strategy – including test methods and test evaluation criteria –is formulated and a test plan is produced. In addition to that test schedule, the framework for quality assurance and test documentation is also constructed.
During detailed design validation support tools are developed or acquired test procedures should be produced. Test cases and test data should be generated. The design itself should be examined for errors. Missing cases, faulty logic, modules interface mismatch, data structure inconsistencies, erroneous I/O assumptions and user interface inadequacies are matter of concern.

Program (Build/Construction):

Actual testing occurs during construction stage of development using various tools and techniques. Static and dynamic analysis too can be used to detect errors. Formal verification and proof techniques provide further quality assurance.

Test Process:

During test process careful control and management of test sets, test results and test reports are carried out and stored in database.

Installation:

It is a process of placing the tested programs into production. Testing during this phase must ensure that the correct versions of the program are placed into production.

Maintenance:

As the system is used it is modified either to correct errors or to augment the original system. After each modification at any level of the system should be retested, re-verified and updated subsequently.

CHECKLISTS: THE VERIFICATION TOOL

An important tool for verification is the checklist, especially in more formal form of verifications like inspections. There are checklists for requirements, functional design specifications, internal design specifications and for test plans. We can also develop our own checklists for anything that we may want to review like code verification checklists, document verification checklists, etc. Checklists are important part of test ware. To get maximum leverage on verification, they should be carefully kept, improved, developed, updated and maintained.

V & V OBJECTIVES

By using verification techniques we can determine that the software system under test performs only the intended functions or in other words verification is necessary to check whether the system performs those functions, which has been specified in the requirements.
By using validation techniques we can ensure that no unwanted or unintended functions are performed which is not specified in the requirements.
Quality: Quality is a characteristic or attribute of any object that refers comparison with known standards. Quality and reliability are incorporated into the software throughout the software process. Proper application of methods and tools, effective formal technical reviews and solid management and measurement all lead to quality and reliability that is confirmed during testing.

VERIFICATION AND VALIDATION EXAMPLES

Verification and validation techniques as discussed in above sections are extremely important for testing. They constitute the basis of software testing. Verification is a quality assurance process while validation is quality control process. The relevance of V & V techniques is furnished by several examples.

VERIFICATION

Verification involves thorough analysis of documents, design structure, or code, which may use informal or formal methods. Using these techniques responsible team members can identify the missing and unspecified requirements, analyze for errors and faults errors etc.

The table below shows those techniques where verification can prove to be vital and effective. It also throws light to the tasks performed by corresponding team member on each technique.

Verification example	Performed by	Explanation	Deliverables
Requirements reviews	Developers, Users	It is the study and discussion of the computer system requirements to ensure they meet stated user needs and are feasible.	They are the reviewed requirements statement ready to be translated into system design.
Design reviews	Developers	It is the study and discussions of the computer system design to ensure it will support the system requirements.	These are system design, ready to be translated into computer programs, hardware configurations, documentation and training.
Code walkthrough	Developers	This contributes to an informal analysis of the program source code to find defects and verify coding techniques.	It is software that is ready for testing or more detailed inspections by the developer.
Code inspections	Developers	It is a formal analysis of the program source code to find defects as defined by meeting system design specifications.	They are software ready for testing by the developer.

VALIDATION

Validation techniques are used to ensure the correctness of the product development. It is accomplished by using different methods at different levels. The table below shows the levels where validation techniques are used along with the tasks performed by each corresponding team member.

Validation example	Performed by	Explanation	Deliverables
Unit Testing	Developers	It includes the testing of a single program, module or unit of a code. It validates that the system performs as per the design.	Here the unit is ready for testing with other system component such as other software units, hardware, documentation or users.
Integration Testing	Developers	It is the testing of related programs, modules, or units of code. Integration testing validates that multiple parts of the system interact according to the system design.	The portions of the system are ready for testing with other portions of the system.

System Testing	Developers, Users	Here the testing of entire computer system, which includes functional and structural for validation of system requirements, takes place.	Here it is a tested computer system, based on what is specified to be developed or purchased.
User Acceptance Testing	Users	The testing of a computer system or parts of a system to make sure it will work in the system regardless of what the system requirements indicate.	Here we have a tested computer system, based on user needs.

REVIEWS

Review involves a meeting of a group of people, whose intention is to evaluate a software related item. Reviews are a type of static testing technique that can be used to evaluate the quality of software artifacts such as requirements document, a test plan, a design document or a code component.

The composition of a review group may consist of managers, clients, developers, testers and other personnel, depending on the type of artifact under review.

REVIEW OBJECTIVES

The general goals for the reviewers are to:

- Identify problem components or components in the software artifact that need improvement.
- Identify problem components in the software artifact that do not need improvement.
- Identify specific errors or defects in the software artifact.
- Ensure that the artifact conforms to organizational standards.

Other review goals are informational, communicational and educational where by review participants learn about the contents of the developing software artifacts, to help them understand the role of their own work and to plan for the future stages of development.

BENEFITS OF REVIEWING

Reviews often represent project milestones and support the establishment of a baseline for a software artifact. Review data can also have an influence on test planning which can help test planners, select effective classes of test and may also have an influence on testing goals.

In some cases, clients attend the review meetings and give feedback to development team so reviews are also a means for client communication. To summarize, the benefits of review programs are

- The quality of the software is higher.
- Productivity (shorter rework time) is increased.
- There is a closer adherence to project schedules (improved process control).
- Awareness of software quality issues can be increased.
- It provides opportunity to identify reusable software artifacts
- Maintenance costs can be reduced.
- More effective test planning can be done
- Customer satisfaction is higher
- A more professional attitude is developed on the part of development staff

Reviews are characterized by its less formal nature compared to inspections. The reviews are more like a peer group discussion with no specific focus on defect identification, data collection and analysis. Reviews do not require elaborate preparation

WALKTHROUGHS

Walkthroughs are a type of technical review where the producer of the review material serves as the review leader and actually guides the progress of the review. They are normally applied on the following documents:

- The design document
- The software code

- The data description
- The reference manuals
- The software specification documents

In the case of detailed design and code walkthroughs, test inputs may be selected and review participants walkthrough the design or code with the set of inputs in a line by line manner. The reader can compare this process to manual execution of the code. If the presenter gives a skilled presentation of the material, the walkthrough participants are able to build a comprehensive mental model of the detailed design or code and are able to evaluate its quality and detect defects.

The primary intent of a walkthrough is to make a team familiarized with a work product. Walkthroughs are useful when one wants to impart a complex work product to a team. For example, the lead designer for the coding team can do a walkthrough for a complex design document.

ADVANTAGES OF WALKTHROUGHS

The following are the advantages of a walkthrough that makes it attractive for reviewing less critical artifacts

- One can eliminate the checklist and preparation step for a walkthrough.
- There is usually no mandatory requirement for a walkthrough, neither a defect list.
- There is also no formal requirement for a follow-up step.

SOFTWARE INSPECTION

Software inspection is a verification process, which is carried out in a formal, systematic and organized manner. Verification techniques are used during all phases of software development life cycle to ensure that the product generation is happening the "right way".

The various verification techniques other than formal inspection are reviews and walkthroughs. These processes work directly on the work product under development. The other verification techniques, which also ensure product quality, are management reviews and audits. We will be focusing more on reviews, walkthroughs and formal inspections.

CHECKLISTS

Checklists are an integral part of successful inspection. They provide the inspectors a focus that is derived from the organizations history of problems to avoid. They serve to educate when the inspectors are first using them. They should evolve as the organization learns about common defects that can be added to the checklists. A checklist that does not change cannot be as useful for improving effectiveness.
Checklists vary throughout the public literature. It is difficult to build a universal checklist that contains all the aspects.

Checklists are used:

- During preparation by inspectors to ensure that the material has been sufficiently examined for inspection meeting.
- At the conclusion of inspection meeting by the moderator to ensure the work product has been sufficiently inspected.

CHARACTERISTICS OF GOOD CHECKLISTS

- Checklists must ultimately be derived from the rules of the process which itself is being checked by inspection.
- Checklists should be kept updated to reflect experience of frequent defects.
- A checklist must concentrate on questions that will turn up major defects.
- Checklists are part of on-the-job training of a checker.
- Checklists should be selective and focus on the key defects possibilities.
- They should be product and domain specific.
- They should be complete and as short as possible.
- They should not get too large to not to get used and not well thought out.
- They need not address every possible defect that can be made.

CHECKLIST TYPES

Checklists can be informally categorized into two types. Work Products and Quality Attributes. Each type can be classified into sub-types, for example:

Work Products

- Requirements
- High- level design

- Low-level design
- Code
- Plans

Quality Attributes
- Usability
- Installability
- Performance
- Maintainability

CHECKLISTS EXAMPLES

The following work product checklists are provided as examples only. The specific checklists to be used by any organization can be obtained initially from these lists and modified over time based on data from practical inspections.

Separate and specific checklists are used for each of different types of inspections. Some of them are given below:

- Requirement specification (RS)
- Architecture (AR)
- High-level design (HLD)
- Low-level design (LLD)
- Code (CD)
- Unit test plan (UTP)
- Unit test case (UTC)
- Function integration test plan (FTP)
- Function integration test case (FTC)
- System integration test plan (STP)
- System integration test case (STC)
- User documentation (DOC)
- Problem Reports (PR)
- Change Requests (CR)

References

For further reading the following books can be referred.

PERRY, WILLIAM E., Effective Methods for Software Testing, Wiley 2000.
PRESSMAN, ROGER S., Software Engineering A Practitioner's Approach, McGraw Hill, 2001.
BURNSTEIN, ILENE, Practical Software Testing, Springer, 2002.
RADICE, RONALD A., Software Inspections, Tata McGraw Hill, 2003.
FREEMAN, DANIEL P. & WEINBERG, GERALD M., Handbook of Walkthroughs, Inspections, and Technical Reviews, Dorset House Publishing, 2003.

MODULE EXERCISES

1. What are the various phases of V model?
2. What are the limitations in walkthrough?
3. What are the benefits of reviewing?
4. What are the validation activities performed by developers?

POINTS TO PONDER

1. Verification and validation are both needed for ensuring quality of the product. Comment.

2. Some issues are better found via inspection while others are better found via testing. State why this may be so. Also state an example of an issue that is easily detected via inspection and another issue that is better found via testing.

3. How do checklists aid in product quality?

MODULE – KEY CONCEPTS

TERMINOLOGY/DEFINITIONS

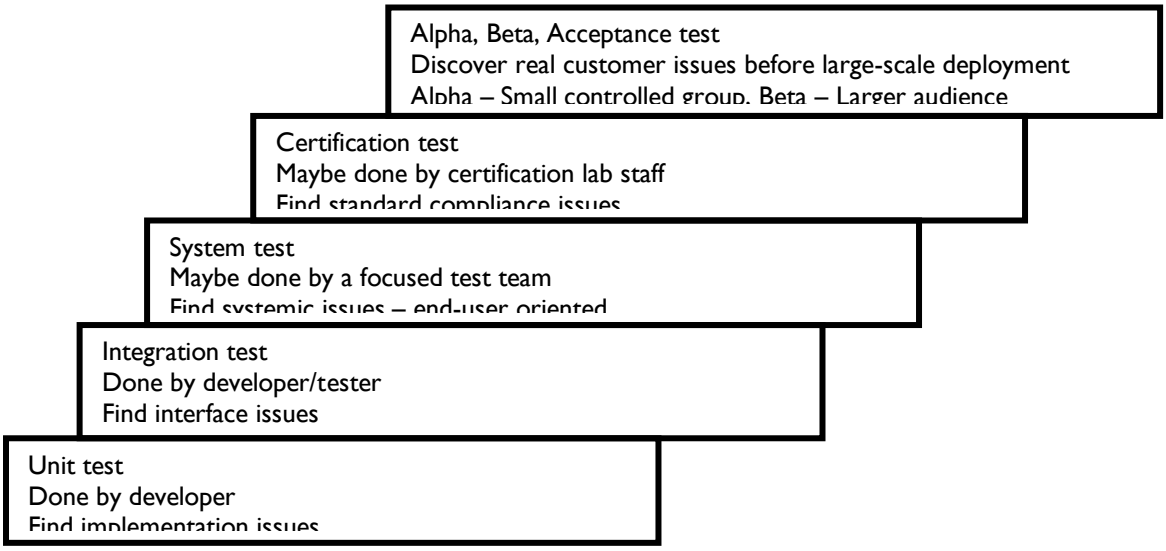
Error	It is a mistake, misconception or misunderstanding on the part of software engineer.
Fault	It is a defect introduced into the software as a result of an error at the input that may cause the system to behave incorrectly and not according to the specification.
Failure	A failure is the inability of the system or component to perform its required functions within specified performance requirement.
Unit testing	It is a level of testing which is done on the smallest functional part of the product.
Integration testing	It is the phase where tester identifies the interface issues between independently developed and unit tested modules.
System testing	It is the phase where the system will be tested as a whole.
System integration testing	System integration testing verifies that a system is integrated to any external or third-party systems defined in the system requirements.
Certification testing	It is done after system testing and focuses on standards compliance aspects of the product if any.
Alpha testing	It is done by users of the system in a controlled environment.
Beta testing	In this testing, the software under test is released to a cross section of users, who will test the software under real world conditions which is nearly impossible to simulate in the testing lab.
Regression testing	It is a test that is run every time when a change is made to the software so that user can find that the change has not broken or altered any other part of the software program.
Functional testing	It refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation.
Non-functional testing	It refers to aspects of the software that may not be related to a specific function or user action, such as scalability or performance under certain constraints, or security.
Static testing	Reviews, walkthroughs, or inspections are considered as static testing.
Dynamic testing	Executing programmed code with a given set of test cases is referred to as dynamic testing.
White-box technique	It is a technique of testing software that tests internal structures or workings of an application. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.
Penetration testing	In penetration testing, white-box testing refers to a methodology where an ethical hacker has full knowledge of the system being attacked. The goal of a white-box penetration test is to simulate a malicious insider who has some knowledge and possibly basic credentials to the target system.
Code coverage	Creating tests to cause all statements, branches and paths in the program to be executed at least once.
Fault injection methods	Improving the coverage of a test by introducing faults to test code paths.
Black-box technique	It is a method of software testing that tests the functionality of an application. Test cases are built around specifications and requirements. These tests can be functional or non-functional. The test designer selects valid and invalid inputs and determines the correct output.
Decision tables	It is a table which associates conditions with actions to perform.
All-pairs testing	It is a combinatorial software testing method that tests all possible discrete combinations of those parameters.
State transition table	It is a table showing what state a finite semi-automaton or finite state machine will move to, based on the current state and other inputs.
Equivalence partitioning	It is a software testing technique that divides the input data of a software unit into partitions of data from which test cases can be derived. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.
Boundary value analysis	Boundary conditions are those situations at the edge of the planned operational limits of the software. Test valid data just inside the boundary, test last possible data and test the invalid data just outside the boundary.
Grey box testing	It involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.
SWEBOK - Software Engineering Body of Knowledge	SWEBOK is a product of the Software Engineering Coordinating Committee sponsored by the IEEE Computer Society.
Destructive testing	Destructive testing attempts to cause the software or a sub-system to fail, in order to test its robustness.
Benchmarks	Allowing run-time performance comparisons to be made.
Performance analysis	It helps to highlight hot spots and resource usage.
Test automation	It is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting

	functions.
End-to-end testing	Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.
Sanity testing	Testing to determine if a new software version is performing well enough to accept it for a major testing effort.
Smoke testing	It is non-exhaustive software testing, ascertaining that the most crucial functions of a program work, but not bothering with finer details.
Ad hoc testing	It is a least formal test method. Ad hoc testing is used to perform without planning and documentation.

TESTING LEVELS

LEVELS OF TESTING

The various levels of testing are as shown below:



Unit Testing:
It is a level of testing, which is done on the smallest functional part of the product. Normally a development team does it.

Integration Testing:
Integration testing is the phase where we identify the interface issues between independently developed and unit tested modules. Developers and testers usually do this.

System Testing:
System testing is the phase where the system will be tested as a whole. The goal is to ensure that the 'system' performs according to the requirements.

Certification Testing:
Certification testing is done after system testing and focuses on standards compliance aspects of the product if any. It is usually done in a lab to check whether the developed software meets the standards specified by various organizations.

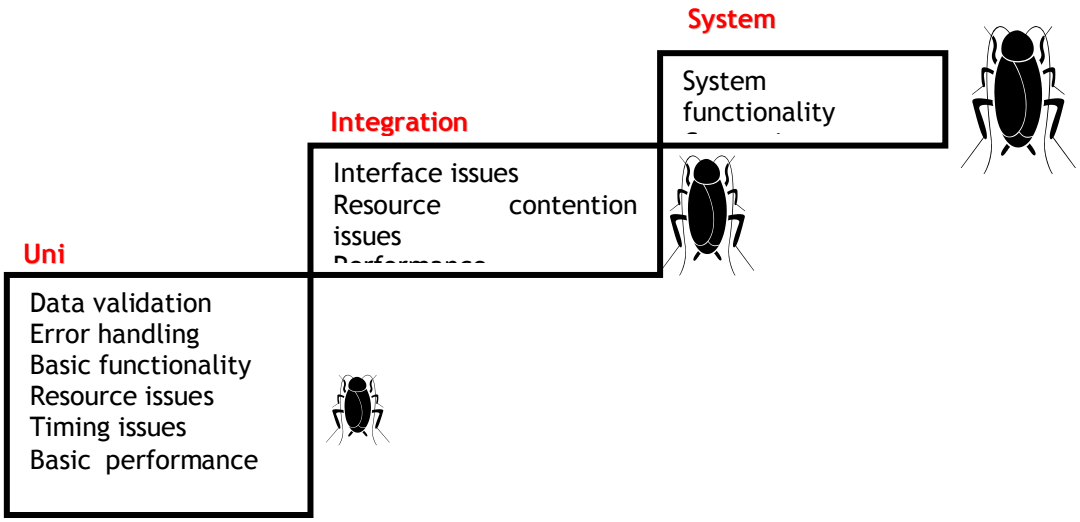
Alpha Testing:
Alpha testing is done by users of the system in a controlled environment. Here, the users will be directly involved with the development team for clarifications.

Beta Testing:
In beta testing, the software under test is released to a cross section of users, who will test the software under real world conditions, which is nearly impossible to simulate in the testing labs.

Acceptance Testing:
Acceptance test is done prior to the installation of the software at the customer's site, which is done, based on the acceptance test plan by the customer. Once the acceptance test is completed, the software can be installed at the customer's premises.

DEFECT TYPES TO BE UNCOVERED AT EACH LEVEL

Unit Testing:
Unit testing, as the name indicates should capture all the defects related to the unit or module being developed. The intention of unit testing is to make sure that when the product is ready for integration testing, the integration testing will not be hampered by unit level defects. It helps in detecting the basic functionality, performance, data validation, error handling, etc.



Integration Testing:
Primary intention of integration testing is to uncover interface related defects between units or modules. It also helps in finding whether the interfaced modules affect the performance of the system, cause any resource consumption and version mismatch issues.

System testing:
System Testing basically deals with the functional and non-functional (constraints) requirements of the system. Testing is carried out to uncover defects and validate whether it meets the software specification of the product.

REGRESSION TESTING

They are tests that are run every time a change is made to the software so that we can find that the change has not broken or altered any other part of the software program. Regression testing is an important strategy for reducing side effects. Regression testing can be done at any level of testing. Regression testing may be conducted manually, by re-executing a subset of all test cases or by using automated capture/playback tools.

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur and new control logic is invoked.

These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, the regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

In a broader context, successful tests results in the discovery of errors and errors must be corrected. Whenever software is corrected some aspect of software configuration like the program, its documentation, etc. is changed. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors.

As the integration testing proceeds the number of regression tests can also grow quite large. Regression testing may be conducted manually or by using automated tools. It is impractical and inefficient to re-execute every test for every program function once change has occurred. Therefore, regression test suite should be designed to include only those tests that address one or more classes of errors in each of the major program functions.

A regression test suite is the subset of tests to be executed. It contains three different classes of test cases:
- A representative sample of tests that will exercise all software functions is one of the classes.
- Additional tests that focus on software functions that are likely to be affected by the change.
- Tests that focus on the software components that have been changed constitute the third type of class.

References:

KIT, EDWARD, *Software Testing In the Real World,* Pearson Education, 2003.
PERRY, WILLIAM E., *Effective Methods for Software Testing,* Wiley 2000.
PRESSMAN, ROGER S., *Software Engineering A Practitioner's Approach,* McGraw Hill, 2001.

MODULE EXERCISES

1. What kinds of issues are found at integration level testing?
2. How can beta testing add value when compared to alpha testing?
3. What are the different classes of test cases that a regression test suite can contain?

POINTS TO PONDER

1. Will Alpha/Beta testing still find defects even if good System testing has been done very well? Explain your answer.
2. If we apply white-box techniques very well, it is not necessary to use black-box techniques. Explain your answer.
3. Despite good unit testing, it is necessary to do system testing. Why?

MODULE – UNDERSTANDING DEFECTS

TERMINOLOGY/DEFINITIONS

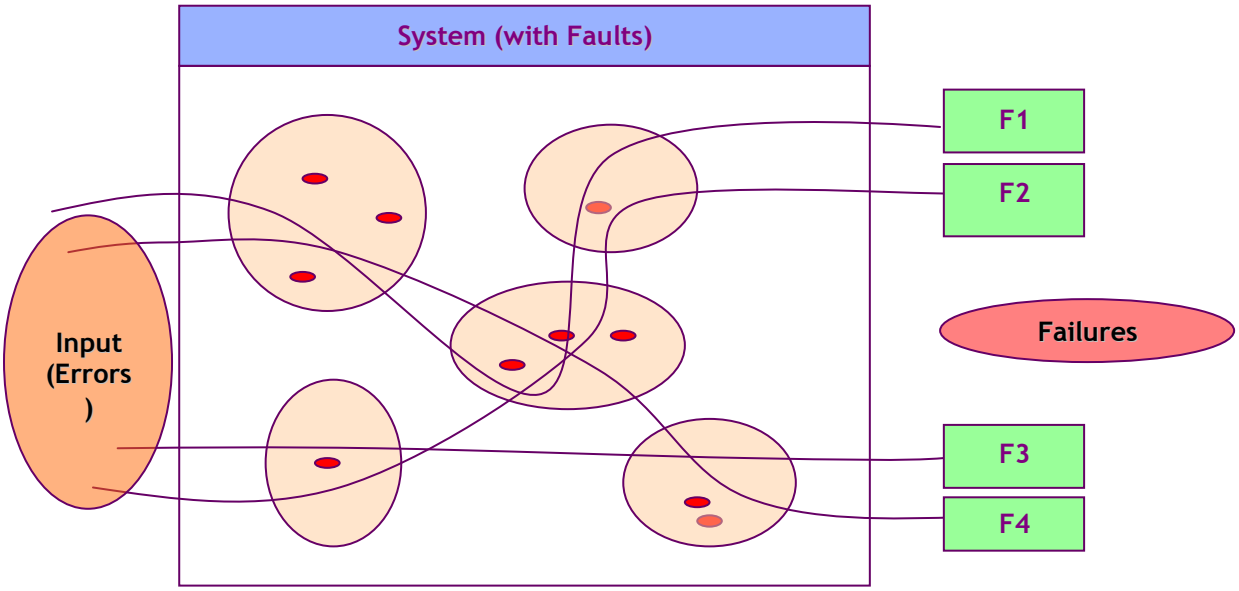
Software bug	It is a term used to describe an error, flaw or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.
Bug ID	A unique identification number which is assigned to a new bug.
Severity	Tells us how bad the defect is. It decides upon the business risk.
Priority	Tells us how soon it is desired to fix the problem. It is based on the software release risk.
Blocker	The bug which prevents one to test the software.
Critical	The bug due to that software crashes, hangs, or causes to lose data.
Major	A bug which causes breakage of major feature of the software.
Minor	A bug which causes minor loss of function of module and there's an easy work around.
Trivial	It is related to a cosmetic problem, such as a misspelled word or misaligned text.
Enhancement	Request for new feature or some enhancement in existing feature of software.

FAULTS IN THE SYSTEM

IEEE Standard Dictionary of Electrical and Electronics Terms (IEEE Standard 100-1992) defines defect as a "product anomaly."

The terms fault and defect are synonymous within the context of software process. Both imply a quality problem that is discovered during the life cycle process. A fault may be injected into a system during any phase of the product development. The origin of faults in a system is discussed in the forthcoming sections.

ERROR, FAULT, FAILURE



Error:
It is a mistake, misconception, or misunderstanding on the part of a software engineer.
When mistakes or errors occur at the input side then they enter into the system as faults/defects, which further lead to the failure of the system if undetected early in the life cycle development process.

Fault:
A fault is a defect introduced into the software as a result of an error at the input that may cause the system to behave incorrectly and not according to the specification. A fault is sometimes called a bug. A fault can occur in software artifacts such as the requirements and design documents.

Failure:
A failure is the inability of the software system or component to perform its required functions within specified performance requirements. During development, a tester normally observes failure and fault is located and repaired by developers. When the software is in operation, users/customers may observe failures, which are reported back to the development team for repairing. A fault in the code does not always produce failure. But some can be catastrophic to the project.

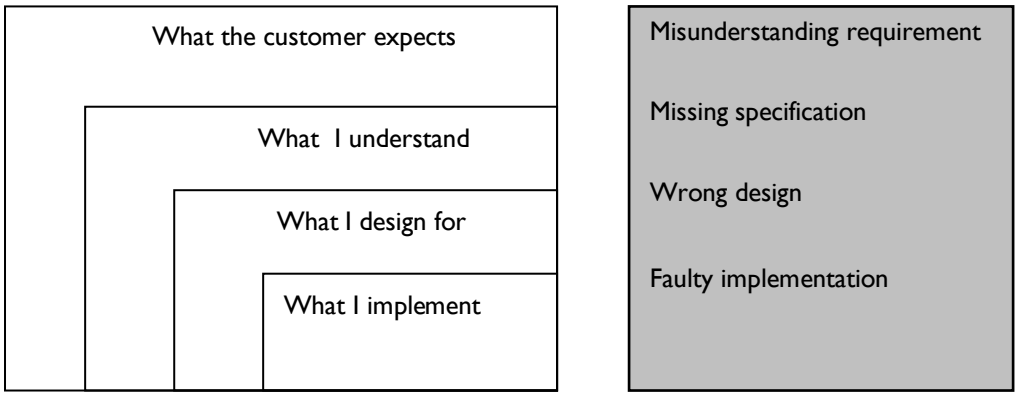
FAULT IN THE SYSTEM

A fault or defect model can be described as link between the error made (e.g., a missing requirement, a misunderstood design element or typographical error) and the fault in the software. Fault can occur during any phase of the software development cycle. It begins with the deviation from customer expectation, passes through wrong design and implementation, and may continue till the post product release process.

In order to keep track of defective areas and for the easy identification, the proposed fault model proves to be very useful and effective.

The fault models are often used to generate a fault list or areas where faults are likely to occur and where we need to focus more. Although the software engineers are not concerned with the physical defects and their relationship between failures, software defects and their origins are not easily mapped. We often use the fault model concept and fault lists accumulated in memory from years of experience to design tests and for the diagnosis tasks during fault debugging activities.

In the past, developers and testers have often used fault model and fault lists in an informal manner to help in developing a defect classification schema and to increase the effectiveness of their testing and debugging processes.



REQUIREMENTS AND SPECIFICATION DEFECTS

The beginning of the software life cycle is critical for ensuring high quality in the software being developed. Defects injected in the early phases can persist and be very difficult to remove in the later phases. Since many requirement documents are written in natural language representation it often happens that they are ambiguous, unclear, redundant, contradictory and imprecise, specification also poses similar problems.

Some specific requirements/specification defects are:

Functional Description Defects
The overall description of what the product does and how it should behave (input/output), is incorrect, incomplete and/or ambiguous.

Feature Defects
Features may be described as distinguishing characteristics of a software component or system. Features refer to functional aspects of the software that map to functional requirements and quality requirements like performance or reliability as described by the user and clients. These defects are due to missing, incorrect, incomplete or superfluous description.

Feature Interaction Defects
These are due to incorrect description of how the features should interact with other systems. For example, suppose one feature of a software system supports adding a new customer to the database. This feature interacts with another feature that categorizes the new customer. The classification feature impacts on where the storage algorithm places the new customer in the database, and also affects another feature that periodically sends advertising information to customers in a specific category.

Interface Description Defects
Interface defects occur in the description of how the target software is to be interfaced with external hardware, software and users.

DESIGN DEFECTS
Design defects occur when system components, interactions between system components, interactions between the components and outside hardware/software, or users are incorrectly designed. This covers the design defects in the algorithms, control, logic, data elements, module interface descriptions and external software/hardware/user interface description. When describing these defects we assume that the detailed description of the software modules is at the pseudo code level with processing steps, data structures, input/output parameters and major control structures defined. Each kind of defect in this phase has been discussed below.

Algorithm and Processing Defects
These defects occur when the processing steps in the algorithm as described by the pseudo code are incorrect. For example, a calculation can be incorrectly specified or a missing step in the code.

Control, Logic and Sequence Defects
Control defects occur when the logic flow in the pseudo code is incorrect. For example, branching too soon, branching too late, or use of incorrect branching condition give rise to above mentioned defects.

Data Defects
These defects are associated with incorrect design of data structures like a record may be lacking field or an incorrect type is assigned to a variable or a field in a record.

Module Interface Description Defects
These are defects derived from, for example, using incorrect, and/or inconsistent parameter types, an incorrect number of parameters, or an incorrect ordering of parameters.

Functional Description Defect
In this category we have incorrect, missing and/or unclear design elements where the design may not properly describe the correct functionality of a module. These defects are best detected during design review.

External Interface Description Defect
These are derived from incorrect design description for interfaces with COTS (Component-Off-The-Shelf) components, external software systems, databases and hardware devices along with user interface description defects where there are missing or improper commands, improper sequence of commands, lack of proper messages and/or lack of feedback messages for the user.

CODING DEFECTS
Coding defects are derived from errors in implementing the code. They are closely related to design defect classes. Some coding defects may come from a failure to understand the programming language constructs and miscommunications with the designer. Others may have transcription or omission origins.

Algorithm and Processing Defects
Adding levels of programming details to design, the coding defects would now include unchecked overflow and underflow conditions, comparing inappropriate data types, converting one data type to another, incorrect ordering of arithmetic operators, misuse or omission of parenthesis, precision loss and incorrect use of signs.

Control, Logic and Sequence Defects
On the coding level these defects would include incorrect expression of case statements, incorrect iteration of loops and missing paths.

Typographical Defects
These are principally syntax errors like incorrect spelling of a variable name.

Initialization Defects
They occur when initialization statements are omitted or are incorrect which happens due to misunderstanding or lack of communication between programmers/designers or carelessness in the programming environment.

Data-flow Defects
There are certain reasonable operational sequences that data should flow through. For example, a variable should be initialized before it is used for calculation or a condition. It should not be initialized twice before use and neither should it be disregarded before it is used.

Data Defects
These defects are indicated by incorrect implementation of data structures like omission of field in a record. It can include flags, indices, and constants set incorrectly.

Module Interface Defects
As in the case of module design elements, interface defects in the code may be due to using incorrect or inconsistent parameter types, an incorrect number of parameters, or improper ordering of the parameters. In addition to defects due to improper design and improper implementation of design, programmers may implement an incorrect sequence of calls or calls to nonexistent modules.

Code Documentation Defects
When the code documentation does not reflect what actually the program does, or is incomplete or ambiguous, this is called code documentation defect. Testers may be misled by documentation defects and thus reuse improper tests or design new tests that are not appropriate for the code.

External Hardware, Software Interface Defects
These defects arise from problems related to system calls, links to databases, input/output sequences, memory usage, resource usage, interrupt and exception handling, data exchanges with hardware, protocols, formats, interface with build files, and timing sequences.

TESTING DEFECTS
Defects are not confined to its code and related artifacts. Test plans, test cases, test harnesses, and test procedures can also contain defects.

Test Harness Defects
In order to test software, especially at the unit and integration levels, auxiliary code must be developed. This is called the test harness or the scaffolding code. The test harness code should be carefully designed and implemented, and tested, since it is a work product and much of this code can be used when new releases are developed.

Test Case Design and Test Procedure Defects
These defects would encompass incorrect, incomplete, missing, inappropriate test cases and test procedures.

References

KIT, EDWARD, *Software Testing In the Real World,* Pearson Education, 2003.
PERRY, WILLIAM E., *Effective Methods for Software Testing,* Wiley 2000.
PRESSMAN, ROGER S., *Software Engineering A Practitioner's Approach,* McGraw Hill, 2001.

MODULE EXERCISES

1. List down the defects/issues that you came across products like Television, DVD player or Mobile Phone when you used them in your day-to-day activities.
2. What was your experience when you approached concerned representatives to resolve any of the above listed issues?

POINTS TO PONDER

1. Whichever experience, the user is not happy with; we should consider it as a defect. Comment.

2. All defects need not be fixed. Comment.

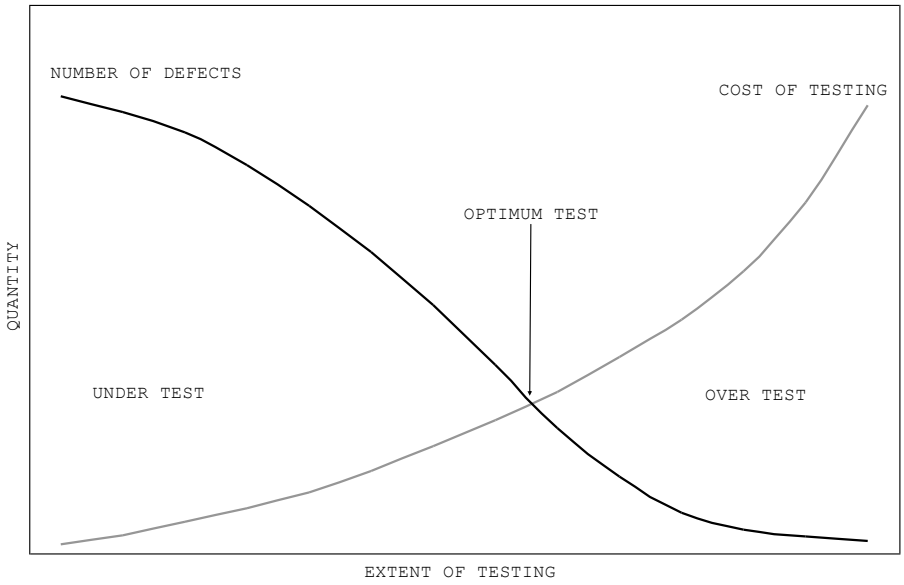
MODULE – RESULTS OF POOR QUALITY/TESTING

ECONOMICS OF TESTING

Software testing is a process that has to be properly balanced so that the economics as well as the project are well under control. Too much of testing results in over-testing and too little testing leads to under-testing. The risk of under-testing points to system defects present in the production environment. The risk of over-testing is the unnecessary use of valuable resources in testing computer systems that have no flaws or so few flaws that the cost of testing far exceeds the value of detecting system defects.

Problems related to testing occur from one of the following causes:

- When we fail to define testing objectives in the testing lifecycle process.
- If testing is carried out at the wrong phase in the testing life cycle.
- Use of ineffective test techniques that lead to unnecessary wastage of time, resources and money.



Cost effectiveness of testing is illustrated in the above figure.

As the cost of testing increases, the number of undetected defects decreases. The left side of the graph represents under-test condition where the cost of testing is less than the resultant loss from undetected defects. The right side represents an over-test condition. In the over-test condition the cost of testing to uncover defects exceeds the losses from those defects. At the point, where the two lines cross, represents the optimum point of testing. A cost effective perspective means testing until the optimum point is reached, which is the point where the cost of testing no longer exceeds the value received from the defects uncovered.

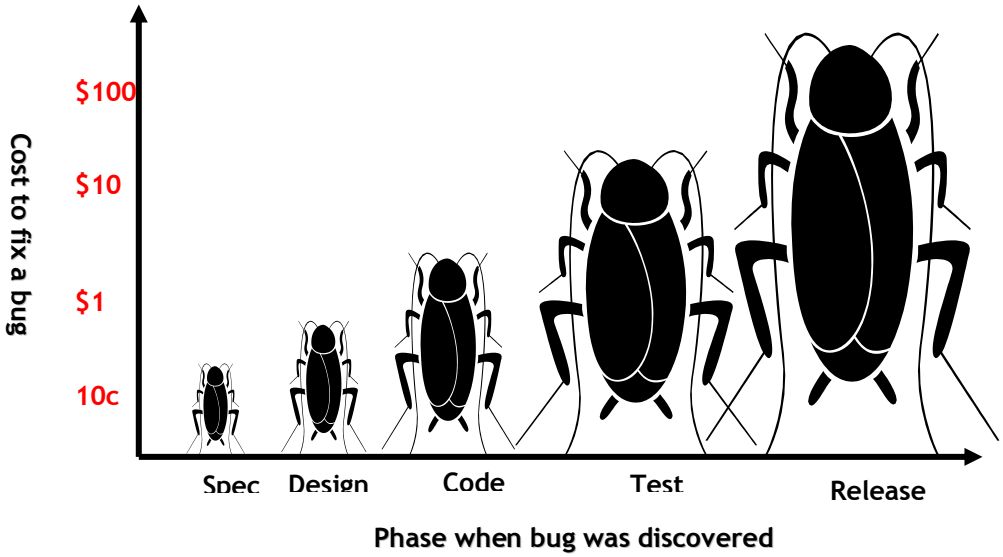
Few organizations have established a basis to measure the testing effectiveness. This makes it difficult for the software programmer/analyst to determine the cost-effectiveness of testing.
Without testing standards, the effectiveness of process cannot be evaluated in sufficient detail to enable the process to be measured and improved.

The use of standardized testing methodology provides the opportunity for a cause and effect relationship to be determined, i.e. the effect of a change in the methodology can be evaluated to determine whether that effect resulted in a smaller or larger number of defects. The establishment of this relationship is an essential step in improving the test process.

COST EFFECTIVE TESTING:

"Too little testing is a crime-too much testing is a sin."
A good tester is always capable of designing better tests that are practical to implement and execute, without approaching exhaustive testing that would never be completed and which prove to be expensive.

Most of the organizations devote 50% of their time in error detection and removal that make use of massive resources which are neither properly documented nor strategically justified. One choice is to be sure we test the right things - to ensure the most critical items are tested and not to waste limited testing resources on unimportant items. Another choice is to test early – to focus on detecting the error closer to the phase where they are introduced rather than testing in the end in order to prevent the costly migration of error's downstream.



For each product, we must implement the most cost effective testing that will ensure that the software is reliable enough, safe enough and meets the customer requirements. Yet another basis of choice of testing is to focus on frequency of use, that means if a part of the system is often used, which has an error, and then its frequent use will increase the chances of failure to higher levels. It is also rational to focus on those areas of the program or system that are more likely to have errors.

The cost of defect identification and correction increases exponentially as the project progresses. The defect encountered during requirement and design phases are cheapest to fix as compared to those found during coding, system testing and installation.

WHY ARE DEFECTS HARD TO FIND?

Finding defects in a system is not easy. Some are easy to spot while others are subtler. Two reasons why defects go undetected:

Not Looking:

Some parts of the system software go untested because developer sometimes assumes that changes don't affect them.

Looking, but not seeing:

Sometimes developers become so familiar with their system that they overlook details, which is why independent verification and validation is used to provide a fresh viewpoint.

Defects typically found in software systems are results of these circumstances:

When IT improperly interprets requirements of the customer.
The users specify the wrong requirements or miss to specify it.
The software engineer sometimes incorrectly records the requirements.
The design specifications are incorrect.
The program specifications and documentation are incorrect.
There are errors in program coding.
There are data entry errors.
There are testing errors.
There are mistakes in error correction.
The corrected condition causes another defect.

HOW DEFECTS AFFECT US - CONSEQUENCES

Defect consequences range from mild to catastrophic. Bugs can be:

Mild

The symptoms of the bug offend us aesthetically like a misspelled output or a misaligned printout.

Moderate

Outputs are misleading or redundant. The bug impacts the system's performance.

Annoying

The system's behavior, because of the bug, is dehumanizing. Names are truncated or arbitrarily modified. Bills for $0.00 are sent.

Disturbing
It refuses to handle legitimate transactions.

Serious
It loses track of transactions; not just the transaction itself, but also the fact that the transaction occurred. Accountability is lost.

Very Serious
Instead of losing your paycheck, the system credits it to another account or converts deposits into withdrawals. The bug causes the system to do the wrong transactions.

Extreme
The problems aren't limited to a few users or to a few transaction types. They are frequent and arbitrary instead of sporadic or for unusual cases.

Intolerable
Long-term, unrecoverable corruption of the database occurs and the corruption is not easily discovered. Serious consideration is given to shutting the system down.

Catastrophic
The decision to shut down is taken out of our hands because the system fails.

Infectious
What can be worse than a failed system? One that corrupts other systems even though it does not fail in itself.

THE IMPORTANCE OF DEFECTS

Defects need to be found out earlier in the software development life cycle which will help in planning and estimating the strategies well in advance as well as in fixing the probable release dates of the deliverables. Defects depend on:

Frequency
How often does a particular kind of bug occur? Need to pay more attention to the more frequent bug types.

Correction Cost
What does it cost to correct the bug after it's been found? That cost is the sum of two factors namely the discovery of the bug and the cost of correction or the fixing of the defect. These costs go up dramatically later in the development cycle when the bug is discovered. Correction cost also depends on system size. The larger the system the more it costs to correct the same bug.

Installation Cost
Installation cost depends on the number of installations: small for a single-user program, but how about a PC operating system bug? Installation cost can dominate all other costs- fixing one simple bug and distributing the fix could exceed the entire system's development cost.

Consequences
What are the consequences of the bug? You might measure this by the mean size of the awards made by juries to the victims of your bug. A reasonable metric for bug importance is:
Importance ($) = frequency * (correction cost + installation cost + consequential cost)

References

ROGER S. PRESSMAN. ,Software Engineering A Practitioner's Approach, McGraw-Hill, 2001
ILENE BURNSTEIN. , Practical Software Testing, Springer, 2002
STEPHEN H. KAN. , Metrics and Models in Software Quality Engineering, Pearson Education, 2002

MODULE EXERCISES

1. Mention the instances where you referred a product/service to friends/family members because you were very satisfied with it.
2. List down the products/services that you rejected because of poor quality.
3. Mention the instances where you were unhappy with the way a product/service worked sometimes.
4. Mention the instances when you suffered any damage due to the products/services that you used misbehaved.

POINTS TO PONDER

1. If we don't discover any more defects, we can be sure that quality is good. Comment.
2. Consider a situation where the software has been tested, but all defects have not been fixed. Would you release this software? If Yes, why? If No, why?
3. Poor quality products/services can take a company out of business. Explain.