SATHYABAMA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Elective: SCSX1060 Software Testing

UNIT 1 – Introduction to Testing

Introduction to Testing

Software testing is a process used to identify the correctness, completeness, and quality of developed computer software. It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users. In other words, software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is defect free.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

Why is testing is important?

Testing is required for an effective performance of software application or product. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development. It's required to stay in the business.

On a quick look we need testing..

1. To discover defects.
2. To avoid user detecting problems
3. To prove that the software has no faults
4. To learn about the reliability of the software.
5. To avoid being sued by customers
6. To ensure that product works as user expected.
7. To stay in business

8. To detect defects early, which helps in reducing the cost of defect fixing?

1. The China Airlines Airbus A300 crashing due to a software bug on April 26, 1994, killing 264 innocent lives. Software bugs can potentially cause monetary and human loss, history is full of such examples.
2. In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
3. In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
4. In may of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars

“Testing is important because software bugs could be expensive or even dangerous”

As Paul Elrich puts it - "To err is human, but to really foul things up you need a computer."

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.
2. It’s essential since it makes sure of the Customer’s reliability and their satisfaction in the application.
3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence.
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It’s important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It’s required to stay in the business.

What is testing?

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product: Meets the business and technical requirements that guided it's design and development.

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product:

- Meets the business and technical requirements that guided it's design and development
- Works as expected
- Can be implemented with the same characteristic.

The definition of Software testing is divided into the following parts:

1) Process: Testing is a process rather than a single activity.

2) All Life Cycle Activities: Testing is a process that's take place throughout the Software Development Life Cycle (SDLC).

- The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it's referred as "verifying the test basis via the test design".
- The test basis includes documents such as the requirements and design specifications.

3) Static Testing: It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example: reviewing, walkthrough, inspection, etc.

4) Dynamic Testing: In dynamic testing the software code is executed to demonstrate the result of running tests. It's done during validation process. For example: unit testing, integration testing, system testing, etc.

5) Planning: We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.

6) Preparation: We need to choose what testing we will do, by selecting test conditions and designing test cases.

7) Evaluation: During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.

8) Software products and related work products: Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

Role of Software Tester

In the planning and preparation phases of the testing, testers should review and contribute to test plans, as well as analyzing, reviewing and assessing requirements and design specifications. They may be involved in or even be the primary people identifying test conditions and creating test designs, test cases, test procedure specifications and test data, and may automate or help to automate the tests. They often set up the test environments or assist system administration and network management staff in doing so. As test execution begins, the number of testers often increases, starting with the work required to implement tests in the test environment. Testers execute and log the tests, evaluate the results and document problems found. They monitor the testing and the test environment, often using tools for this task, and often gather performance metrics. Throughout the testing life cycle, they review each other's work, including test specifications, defect reports and test results.

A Software tester (software test engineer) should be capable of designing test suites and should have the ability to understand usability issues. Such a tester is expected to have sound knowledge of software test design and test execution methodologies. It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently. The roles and responsibilities for a usability software tester are as follows:

1. A Software Tester is responsible for designing testing scenarios for usability testing.
2. He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.
3. He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
4. Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.

A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing. He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects. In order to create test cases it is important that the software tester is aware of various testing techniques and which approach is best for a particular system. He should know what are various phases of software testing and how testing should be carried out in each phase. The responsibilities of the software tester include:

1. Creation of test designs, test processes, test cases and test data.
2. Carry out testing as per the defined procedures.
3. Participate in walkthroughs of testing procedures.
4. Prepare all reports related to software testing carried out.
5. Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Quality

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally develop software

Three (3) important points to remember on software quality

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. There is a set of implicit requirements that often goes unmentioned (e.g., the desire for good maintainability). If software conforms to its explicit requirements, but fails to meet implicit requirements.

Testing and Quality

The Role of Software Testing is often misunderstood across the different stake holders of the application development & this list includes testers too.

Testing is considered to be part of Quality Assurance activity since the initial days of Software Development and the same trend is happening as of now too. Even most of the titles like QA Engineer / QA Lead are associated with Testers even though they are not performing the role of QA.

It's good to capture the mission of Testing & align your test teams in that direction. Everyone in the team must be clear on his / her role and see how the same is helping to achieve the mission of the team.

Many people and organizations are confused about the difference between quality assurance (QA), quality control (QC), and testing. They are closely related, but they are different concepts. Since all three are necessary to effectively manage the risks of developing and maintaining software, it is important for software managers to understand the differences. They are defined below:

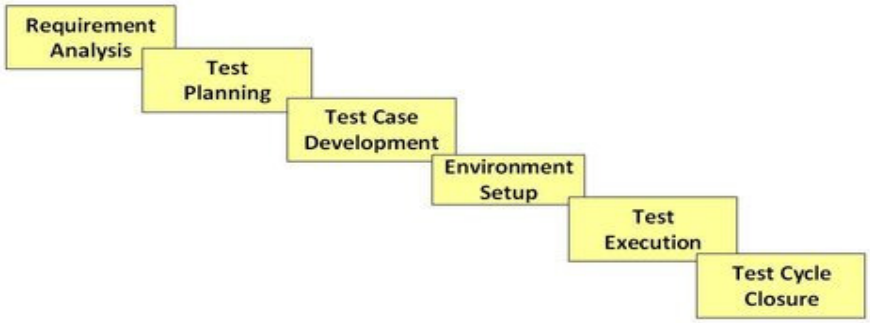
- Quality Assurance: A set of activities designed to ensure that the development and/or maintenance process is adequate to ensure a system will meet its objectives.
- Quality Control: A set of activities designed to evaluate a developed work product.
- Testing: The process of executing a system with the intent of finding defects. (Note that the "process of executing a system" includes test planning prior to the execution of the test cases.)

QA activities ensure that the process is defined and appropriate. Methodology and standards development are examples of QA activities. A QA review would focus on the process elements of a project - e.g., are requirements being defined at the proper level of detail. In contrast, QC activities focus on finding defects in specific deliverables - e.g., are the defined requirements the right requirements. Testing is one example

of a QC activity, but there are others such as inspections. Both QA and QC activities are generally required for successful software development.

Software Testing Life Cycle (STLC)

Contrary to popular belief, Software Testing is not a just a single activity. It consists of series of activities carried out methodologically to help certify your software product. These activities (stages) constitute the Software Testing Life Cycle (STLC). The different stages in Software Test Life Cycle -



Test process

Testing is a process rather than a single activity. This process starts from test planning then designing test cases, preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps:

1) Planning and Control
2) Analysis and Design
3) Implementation and Execution
4) Evaluating exit criteria and Reporting
5) Test Closure activities

Test Information Flow

Two classes of input are provided to the test process:

A software configuration : Software Requirements Specification, Design Specification and Source code

A test configuration : Test Plan, Test Procedure and Testing tools

STLC deliverables

Each of STLC stages have a definite Entry and Exit criteria, Activities & Deliverables associated with it.

	Phase Name	Entry Criteria	Activities Performed	Deliverables
1	Requirements	Requirements specification document	Do brainstorming of the requirements. Create a list of requirements and get	RUD Requirements understanding document.

	Phase Name	Entry Criteria	Activities Performed	Deliverables
		Application design document User acceptance criteria document	your doubts clarified. Understand the feasibility of the requirements whether it is testable or not. If your project requires automation, do the automation feasibility study.	Testing feasibility report Automation feasibility report.
2	Planning	Updated requirements document. Test feasibility reports “ Automation feasibility report.	Define the scope of the project Do the risk analysis and prepare the risk mitigation plan. Perform test estimation. Determine the overall testing strategy and process. Identify the tools and resources and check for any training needs. Identify the environment.	Test Plan document. Risk mitigation document. Test estimation document.
3	Analysis	Updated requirements document Test Plan document Risk Document Test estimation document	Identify the detailed test conditions	Test conditions document.
4	Design	Updated requirements document Test conditions document	Detail out the test condition. Identify the test data Create the traceability metrics	Detailed test condition document Requirement traceability metrics Test coverage metrics

	Phase Name	Entry Criteria	Activities Performed	Deliverables
5	Implementation	Detailed test condition document	Create and review the test cases. Create and review the automation scripts. Identify the candidate test cases for regression and automation. Identify / create the test data Take sign off of the test cases and scripts.	Test cases Test scripts Test data
6	Execution	Test cases Test scripts	Execute the test cases Log bugs / defects in case of discrepancy Report the status	Test execution report Defect report Test log and Defect log Updated requirement traceability metrics
7	Conclusion	Updated test cases with results Test closure conditions	Provide the accurate figures and result of testing Identify the risks which are mitigated	Updated traceability metrics Test summary report Updated risk management report
8	Closure	Test closure condition Test summary report	Do the retrospective meting and understand the lessons learnt	Lessons learnt document Test matrices Test closure report.

Test Maturity Model (TMM)

There is a demand for software of high quality with low defects; process is important in the software engineering discipline; software testing is an important software development sub process; existing software evaluation and improvement models have not ad-equately addressed testing issues.

An introduction to the Testing Maturity Model is now presented as a framework for discussion of these issues, and as a means for addressing them. The focus of the TMM is on testing as a process in itself that can be evaluated and improved. In the testing domain possible benefits of test process improvement are the following:

- smarter testers
- higher quality software
- the ability to meet budget and scheduling goals
- improved planning
- the ability to meet quantifiable testing goals

Test process improvement is supported by the set of levels and maturity goals in the TMM. Achievement of the maturity goals results in incremental improvement of an organization's testing process. The TMM Assessment Model supports test process evaluation.

The development of version 1.0 of the TMM was guided by the work done on the Capability Maturity Model for software (CMM), a process improvement model that has received widespread support from the soft-ware industry in the United States. The CMM is classified architecturally as staged process improvement model. This type of process improvement model architecture prescribes the stages that an organization must proceed through in an orderly fashion to improve its software development process. Other process improvement models can be described as having a continuous type of architecture, for example, the SPICE model. In this type of architecture there is no fixed set of levels or stages to proceed through. An organization applying a continuous model can select areas for improvement from many different categories.

The CMM has five levels or stages that describe an evolutionary pattern of software process maturity and serve as a guide for improvement. Each level has a set of Key Process Areas (KPA) that an organization needs to focus on to achieve maturity at that level. There are also key practices associated with each level that provide support for implementing improvements at that level. The CMM also has an assessment procedure that allows an organization to evaluate the current state of its software process and identify process strengths and weaknesses.

TMM Levels

As in the case of the CMM, the TMM also follows what is called a staged architecture for process improvement models. It contains stages or levels through which an organization passes as its testing process evolves from one that is ad hoc and unmanaged to one that is managed, defined, measured, and optimizable. The internal structure of the TMM is rich in testing practices that can be learned and applied in a systematic way to support a quality testing process that improves in incremental steps. There are five levels in the TMM that prescribe a maturity hierarchy and an evolutionary path to test process improvement. The characteristics of each level are described in terms of testing capability organizational goals and roles/responsibilities for the key players in the testing process, the man-agers, developers/testers, and users/clients.

Each level with the exception of level 1 has a structure that consists of the following:

- *A set of maturity goals.* The maturity goals identify testing improve-ment goals that must be addressed in order to achieve maturity at that level. To be placed at a level, an organization must satisfy the maturity goals at that level.
- *Supporting maturity subgoals.* They define the scope, boundaries and needed accomplishments for a particular level.
- *Activities, tasks and responsibilities (ATR).* The ATRs address im-plementation and organizational adaptation issues at each TMM level. Supporting activities and tasks are identified, and responsibilities are assigned to appropriate groups

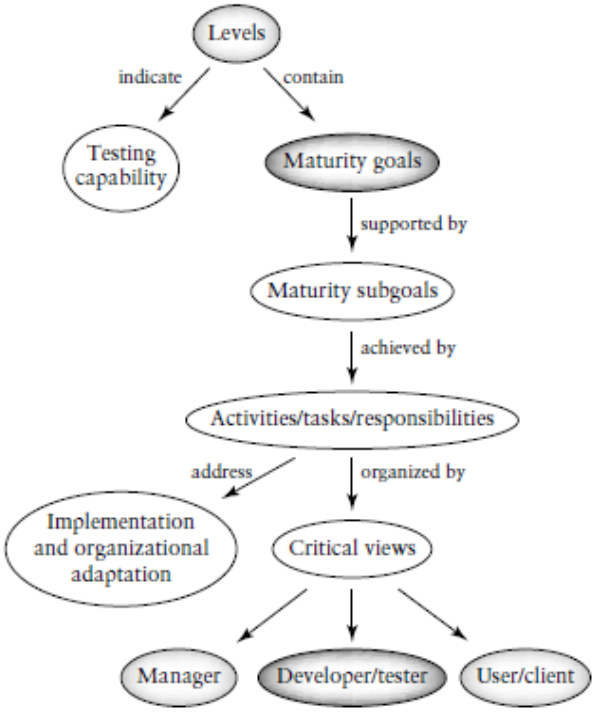


Fig . The internal structure of TMMmaturity levels.

Figure above illustrates the TMM level structure. Each maturity goal at each TMM level is supported by a set of maturity sub goals. The maturity sub goals are achieved through a group of activities and tasks with responsibilities (ATR). Activities and tasks are defined in terms of actions that must be performed at a given level to improve testing capability; they are linked to organizational commitments. Responsibilities are assigned for these activities and tasks to three groups that TMM developers believe represent the key participants in the testing process: managers, developers/testers, and users/clients. In the model they are referred to as "the three critical views (CV)." Definition of their roles is essential in developing a maturity framework. The manager's view involves commitment and ability to perform activities and tasks related to improving testing capability. The developer/tester's view encompasses the technical activities and tasks that, when applied, constitute quality testing practices. The user's or client's view is defined as a cooperating, or supporting, view. The developers/testers work with client/user groups on quality-related activities and tasks that concern user-oriented needs. The focus is on soliciting client/user support, consensus, and participation in activities such as requirements analysis, usability testing, and acceptance test planning.

The maturity goals at each level of the TMM are shown in Figure 1.5. They are fully described in published papers and are also listed below along with a brief description of the characteristics of an organization at each TMM level [2–6]. The description will introduce the reader to the evolutionary path prescribed in the TMM for test process improvement. Additional details are provided in subsequent text chapters.

Level 1—Initial: (No maturity goals): At TMM level 1, testing is a chaotic process; it is ill-defined, and not distinguished from debugging. A documented set of specifications for software behavior often does not exist. Tests are developed in an ad hoc way after coding is completed. Testing and debugging are interleaved to get the bugs out of the software. Software products are often released without quality assurance. There is a lack of resources, tools and properly trained staff. This type of organization would be at level 1 of the CMM.

Level 2—Phase Definition: (Goal 1: Develop testing and debugging goals; Goal 2: Initiate a testing planning process; Goal 3: Institutionalize basic testing techniques and methods)

At level 2 of the TMM testing is separated from debugging and is defined as a phase that follows coding. It is a planned activity; however, test planning at level 2 may occur after coding for reasons related to the immaturity of the testing process. For example, there may be the perception at level 2, that all testing is execution based and dependent on the code; therefore, it should be planned only when the code is complete.

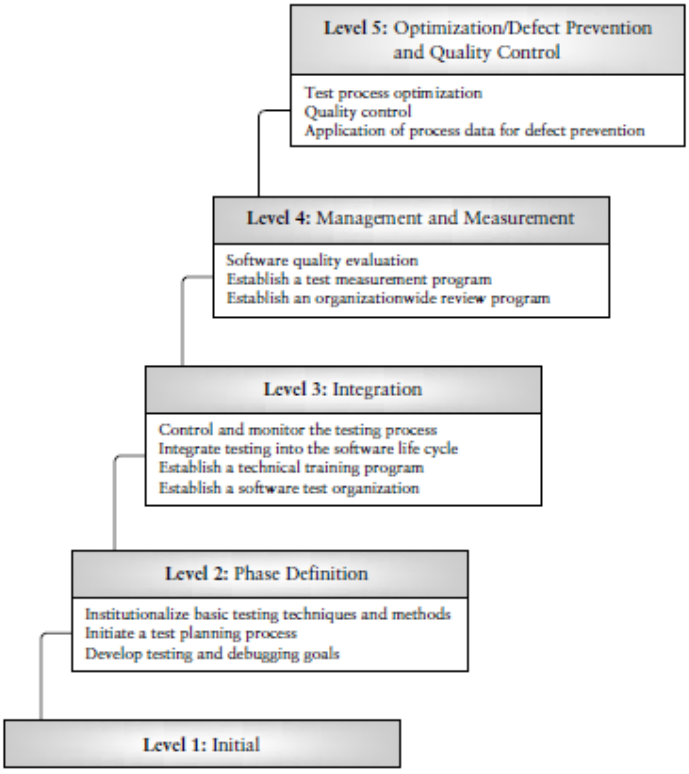


Fig.The 5-level structure of the testing maturity model.

The primary goal of testing at this level of maturity is to show that the software meets its stated specifications [2,5]. Basic testing techniquesand methods are in place; for example, use of black box and white box testing strategies, and a validation cross-reference matrix. Testing is multi-leveled: there are unit,

integration, system, and acceptance levels. Many quality problems at this TMM level occur because test planning occurs late in the software life cycle. In addition, defects are propagated from the requirements and design phases into the code. There are no review programs as yet to address this important issue. Postcode, execution-based testing is still considered the primary testing activity.

Level 3—Integration: (Goal 1: Establish a software test organization; Goal 2: Establish a technical training program; Goal 3: Integrate testing into the software life cycle; Goal 4: Control and monitor testing)

At TMM level 3, testing is no longer a phase that follows coding, but is integrated into the entire software life cycle. Organizations can build on the test planning skills they have acquired at level 2. Unlike level 2, planning for testing at TMM level 3 begins at the requirements phase and continues throughout the life cycle supported by a version of the V-model. Test objectives are established with respect to the requirements based on user/client needs, and are used for test case design. There is a test organization, and testing is recognized as a professional activity. There is a technical training organization with a testing focus. Testing is monitored to ensure it is going according to plan and actions can be taken if deviations occur. Basic tools support key testing activities, and the testing process is visible in the organization. Although organizations at this level begin to realize the important role of reviews in quality control, there is no formal review program and reviews do not as yet take place across the life cycle. A formal test measurement program has not yet been established to quantify a significant number of process and product attributes.

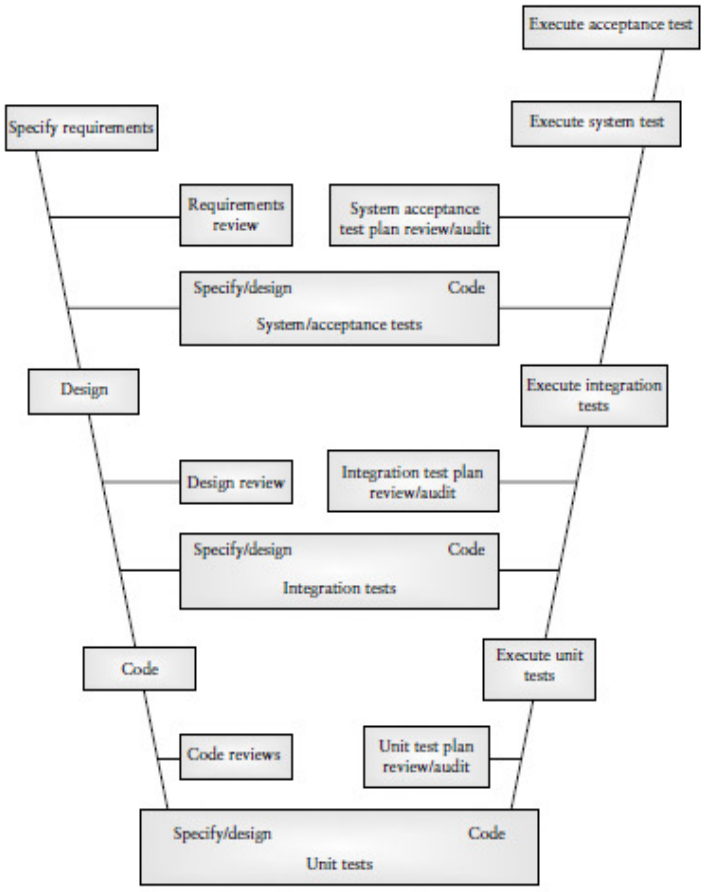
Level 4—Management and Measurement: (Goal 1: Establish an organization wide review program; Goal 2: Establish a test measurement pro-gram; Goal 3: Software quality evaluation)

Testing at level 4 becomes a process that is measured and quantified. Reviews at all phases of the development process are now recognized as testing/quality control activities. They are a compliment to execution-based tests to detect defects and to evaluate and improve software quality. An extension of the V-model as shown in Figure below can be used to sup-port the implementation of this goal. Software products are tested for quality attributes such as reliability, usability, and maintainability. Test cases from all projects are collected and recorded in a test case database for the purpose of test case reuse and regression testing. Defects are logged and given a severity level. Some of the deficiencies occurring in the test process are due to the lack of a defect prevention philosophy, and the porosity of automated support for the collection, analysis, and dissemination of test-related metrics.

Level 5—Optimization/Defect Prevention/Quality Control: (Goal 1: Defect prevention; Goal 2: Quality control; Goal 3: Test process optimization)

Because of the infrastructure that is in place through achievement of the maturity goals at levels 1–4 of the TMM, the testing process is now said to be defined and managed; its cost and effectiveness can be monitored. At level 5, mechanisms are in place so that testing can be fine-tuned and continuously improved. Defect prevention and quality control are prac-ticed. Statistical sampling, measurements of confidence levels, trustwor-thiness, and reliability drive the testing process. Automated tools totally support the running and rerunning of test cases. Tools also provide sup-port for test case design, maintenance of test-related items, and defect collection and analysis. The collection and analysis of test-

related metrics also has tool support. Process reuse is also a practice at TMM level 5 supported by a Process Asset Library (PAL).



The Extended/Modified V-model.

Software Testing Terminologies

Error

An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of developer we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a de-sign notation, or a programmer might type a variable name incorrectly.

Faults (Defect)

A fault (defect) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. Faults or defects are sometimes called “bugs

Failures

A failure is the inability of a software system or component to perform its required functions within specified performance requirements. During execution of a software component or system, a tester, developer, or user observes that it does not produce the expected results. In some cases a particular type of misbehavior indicates a certain type of fault is

TestCase

The tester bundles this information into an item called a test case. A test case in a practical sense is a test-related item which contains the following information:

1. ***A set of test inputs****.* These are data items received from an external source by the code under test. The external source can be hardware, software, or human.

2. ***Execution conditions.*** These are conditions required for running the test, for example, a certain state of a database, or a configuration of a hardware device.

3 ***.Expected outputs****.* These are the specified results to be produced by the code under test

Structural Testing

Structural testing, also known as glass box testing or white box testing is an approach where the tests are derived from the knowledge of the software's structure or internal implementation. The other names of structural testing include clear box testing, open box testing, logic driven testing or path driven testing.

Functional Testing

Functional testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test.

Software Workbench

A Workbench is a method of documenting how a particular activity must be fulfilled. A workbench is referred to a stages, steps, and assignments.

There are five assignments for each workbench:

1. Input: Each task requires certain input and output parameters. For each workbench we need specific inputs. Input form is the first stage of the workbench.
2. Perform: The main goal of the workbench is to change the input forms on the expected output ones.
3. Check: Check ensures that output after the performing achieves the desired result.
4. Production output: If the check is done correctly the production output becomes the last stage of the workbench.
5. Rework: If the result after the check doesn't meet our expectations we need to start again from the step of performance.

In fact, scenarios are not made of one workbench but of many related workbenches. A workbench gives you an opportunity to execute any one task with appropriate software testing.

Eight Considerations for Developing Testing Methodologies

The eight considerations listed below provide the framework for developing testing tactics. Each is described in the following sections.

1. Acquire and study the test strategy
2. Determine the type of development project
3. Determine the type of software system
4. Determine the project scope
5. Identify the tactical risks
6. Determine when testing should occur
7. Build the tactical test plan
8. Build the unit test plans

Acquire and Study the Test Strategy

A team familiar with the business risks associated with the software normally develops the test strategy, and the test team develops the tactics. Thus, the test team needs to acquire and study the test strategy, focusing on the following questions:

- What is the relationship of importance among the test factors?
- Which of the high-level risks are the most significant?
- Who has the best understanding of the impact of the identified business risks?
- What damage can be done to the business if the software fails to perform correctly?
- What damage can be done to the business if the software is not completed on time?

Determine the Type of Development Project

The type of project refers to the environment in which the software will be developed, and the methodology used. Changes to the environment also change the testing risk. For example, the risks associated with a traditional development effort are different from the risks associated with off-the shelf purchased software. Table below illustrates characteristics and testing tactics that can be used for different types of projects.

Project Type	Characteristics	Test Tactics
Traditional system development (and most perfective maintenance)	Uses a system development methodology User knows requirements Development determines structure	Test at end of each task, step and phase Verify that specs match need Test function and structure
Iterative development, prototyping, CASE	Requirements unknown Structure predefined	Verify that CASE tools are used properly Test functionality
System maintenance	Modify structure	Test structure Works best with release methods Requires regression testing
Purchased or contracted software	Structure unknown May contain defects Functionality defined in user documentation Documentation may vary from software	Test functionality Verify functionality matches need Test fit into environment

Table: Characteristics and test tactics of various software projects

Determine the Type of Software System

The type of software system refers to the processing that will be performed by that system. There are sixteen different software system types; however, a single software system may incorporate more than one of these types. Identifying the specific combinations of software making up the project can help analyze lessons learned on past projects with similar types of software.

Determine the Project Scope

The project scope refers to the totality of activities to be incorporated into the software system being tested – the range of system requirements and specifications to be understood. The scope of the testing effort is usually defined by the scope of the project. New system development has a much different scope from modifications to an existing system. When defining the scope, consider the following characteristics and then expand the list to encompass the requirements of the specific software system being tested.

New Systems Development

- Automating manual business process?
- Which business processes will or won't be affected?
- Which business areas will or won't be affected?
- Interfacing to existing systems?
- Existing systems will or won't be affected?

Changes to Existing Systems

- Corrective only?
- Maintenance re-engineering standards?
- Correction to known latent defects in addition to enhancements?
- Other systems affected?
- Risk of regression?

Identify the Tactical Risks

Strategic risks are the high-level business risks faced by the software system. They are decomposed into tactical risks to assist in creating the test scenarios that will address those risks. It is difficult to create test scenarios for high-level risks.

Tactical risks are divided into three categories:
Structural Risks: These risks are associated with the application and the methods used to build it.
Technical Risks: These risks are associated with the technology used to build and operate the application.
Size risks: These risks are associated with the magnitude in all aspects of the software.

Determine When Testing Should Occur

The previous steps have identified the type of development project, the type of software system, the type of testing, the project scope, and the tactical risks. That information should be used to determine the point in the development process at which testing should occur.For new development projects, testing can, and should, occur throughout the phases of a project. For modifications to existing systems, any or all of these may be applicable, depending on the scope. Examples of test activities to be performed during these phases are:

Requirements Phase Activities

- Determine test strategy
- Determine adequacy of requirements
- Generate functional test conditions

- Design Phase Activities
- Determine consistency of design with requirements
- Determine adequacy of design
- Determine adequacy of the test plans
- Generate structural and functional test conditions Program (Build) Phase Activities
- Determine consistency with design
- Determine adequacy of implementation
- Generate structural and functional test conditions for modules and units

Test Phase Activities

- Test application system
- Installation Phase Activities
- Place tested system into production

Maintenance Phase Activities

- Modify and retest

Build the System Test Plan

Using information from the prior steps, develop a System Test Plan to describe the testing that will occur. This plan will provide background information on the system being tested, test objectives and risks, the business functions to be tested, and the specific tests to be performed. The Test Plan is the road map that will be followed when conducting testing. The plan is then decomposed into specific tests and lower-level plans. After execution, the results from the specific tests are rolled up to produce a Test Report.

Build the Unit Test Plans

During internal design, the system is divided into the components or units that perform the detailed processing. Each of these units should have an individual Test Plan. The plans can be as simple or as complex as the organization requires based on its quality expectations.

The importance of a Unit Test Plan is to determine when unit testing is complete. It is not cost effective to submit units that contain defects to higher levels of testing. The extra effort spent in developing Unit Test Plans, testing units, and assuring that units are defect free prior to integration testing can have a significant payback in reducing overall test costs.

VERIFICATION vs VALIDATION

The terms ‘Verification‘and ‘Validation‘are frequently used in the software testing world but the meaning of these terms are mostly vague and debatable.

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business

	Requirements for that phase.	Requirements.
Objective	To ensure that the product is being built according to the requirements And design specifications. In other words, to ensure that work products Meet their specified requirements.	To ensure that the product actually meets the user's needs, and that the specifications were correct in the first Place. In other words, to demonstrate that the product fulfils its intended use when placed in its intended Environment.
Question	Are we building the product *right*?	Are we building the *right* product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	Reviews, walkthroughs, inspections	Testing

Software testability checklist

Operability: if it works better it can be tested more efficiently

Observability : what you see is what you test

Controllability: if software can be controlled better then it is more that testing can be automated and optimized

The following list helps

- Identify the different end users of the system and their interaction with the same.
- Capture the important scenarios for each end user. It's good to note that we need to capture the story around the scenario and not just steps.
- Talk to different stake holders including the customers (incase if you have access) on how the feature might be used & capture the scenarios.
- Plan towards uncovering the critical issues of the system as early as possible.

REFERENCES

1. Ilene Burnstein, Practical software testing : a process-oriented approach, Springer Professional computing Publications, 2002
2. http://istqbexamcertification.com/
3. https://en.wikipedia.org/wiki/Software_testing
4. www.test-institute.org/Software_Testing_Roles_And_Responsibilities.php