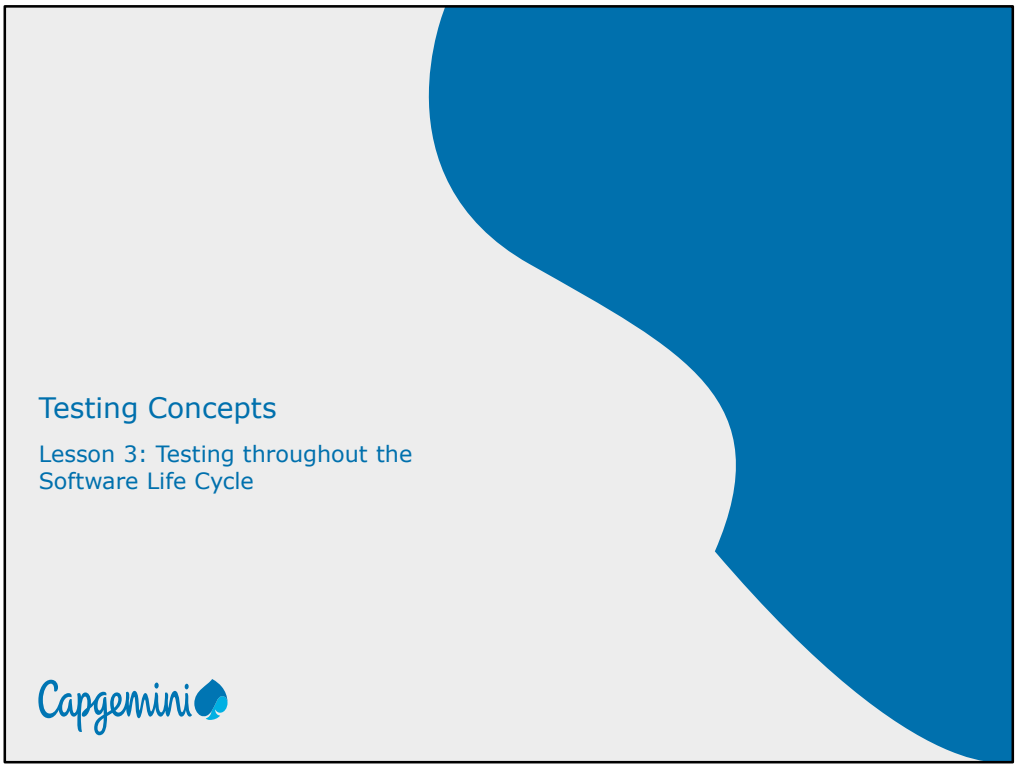

Testing Concepts
Lesson 3: Testing throughout the Software Life Cycle
Capgemini

Lesson Objectives

To understand the following topics:

- Testing throughout the Software Life Cycle
- Introduction of SDLC and V-Model
- SDLC and V-Model
- Iterative Life Cycles
- Rapid Application Development
- Rational Unified Process (RUP) Phases
- RUP Phases and Disciplines
- Agile Development – Extreme Programming (XP)
- Testing Phases
- Introduction of Component Testing
- Component /Unit Testing

Lesson Objectives

To understand the following topics:

- Introduction of Integration testing
- Why Integration Testing is Required?
- Types of Integration testing
- Top Down Integration Testing
- Top Down Integration Testing
- Bottom Up Integration Testing
- Top Down vs. Bottom Up Testing
- Introduction to System Testing
- Types of System Testing

Testing throughout the Software Life Cycle

Testing is not a stand-alone activity

It has its place within a SDLC model

In any SDLC model, a part of testing is focused on Verification and a part is focused on Validation

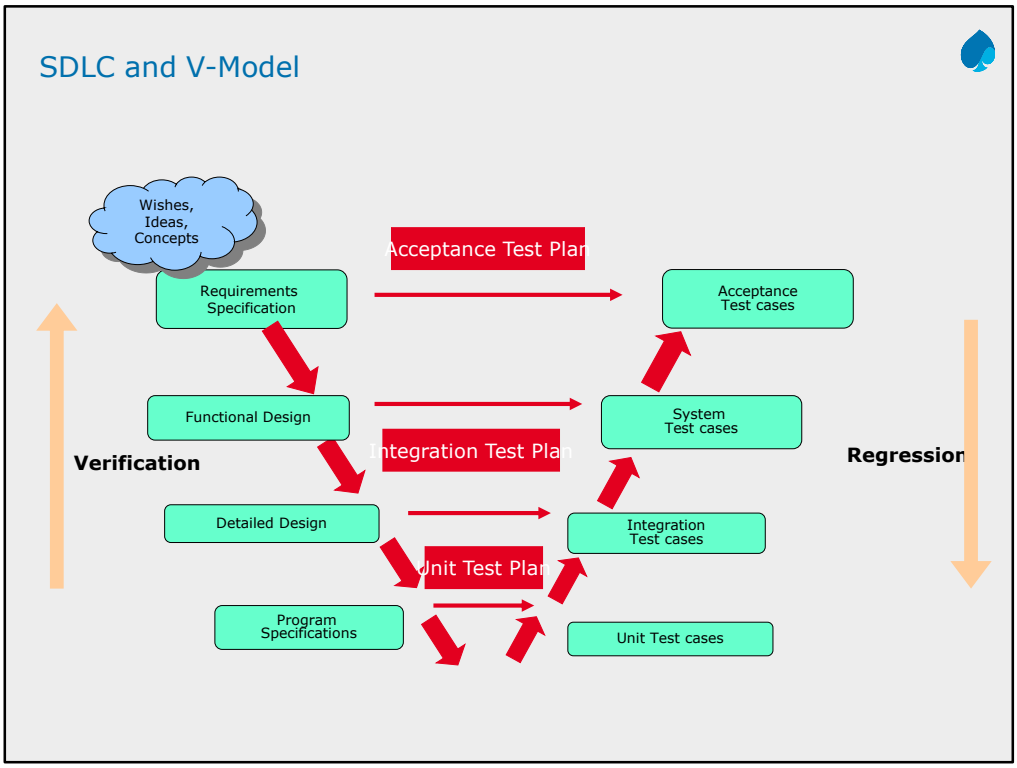
- Verification: Is the deliverable built according to the specification?
- Validation: Is the deliverable fit for purpose?

Various SDLC models are :

- V-Model
- Iterative life cycles
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)
  - Dynamic System Development Methodology [DSDM]
  - Agile - Extreme Programming (XP)

Introduction of SDLC and V-Model

- There are some distinct test phases that take place in each of the software life cycle activity
- It is easier to visualize through the famous Waterfall development model and V- model of testing
- The V proceeds from left to right, depicting the basic sequence of development and testing activities
- The V model is valuable because it highlights the existence of several levels or phases of testing and depicts the way each relates to a different development phase.



V Model
Left side shows activities apart from testing
Right side shows Testing activities
Specific Testing activities are carried in parallel to development activities

Iterative Life Cycles

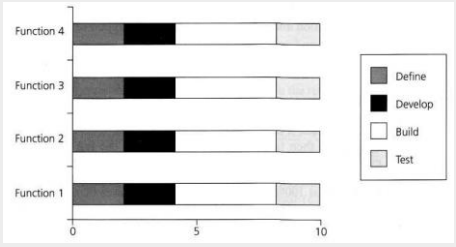
- Iterative life cycle can give early market presence with critical functionality .
- The delivery is divided into increments. Each increment adds new functionality.
- Testing of the new functionality, regression testing, and integration testing are prominent test types performed in iterative life cycle models.
- Simpler to manage because the workload is divided into smaller pieces
- Iterative models are also known as incremental development models

Examples are

- Rapid Application Development (RAD)
- Rational Unified Process (RUP)
- Agile development

Rapid Application Development

- Components are developed in parallel
- The developments are time-boxed, delivered, and then assembled into a working prototype
- Very quickly gives the customer something to see and use
- Rapid development and rapid response to changing requirements is possible
- Allows early validation of technology risks



Rational Unified Process (RUP) Phases

RUP is created by the Rational Software Corporation consisting of four phases and nine disciplines

Inception

- The objective is to define scope the system. The business case which includes business context, success factors and financial forecast is established.

Elaboration

- The objective is to mitigate the key risk items. The problem domain analysis is made and the architecture of the project gets its basic form.

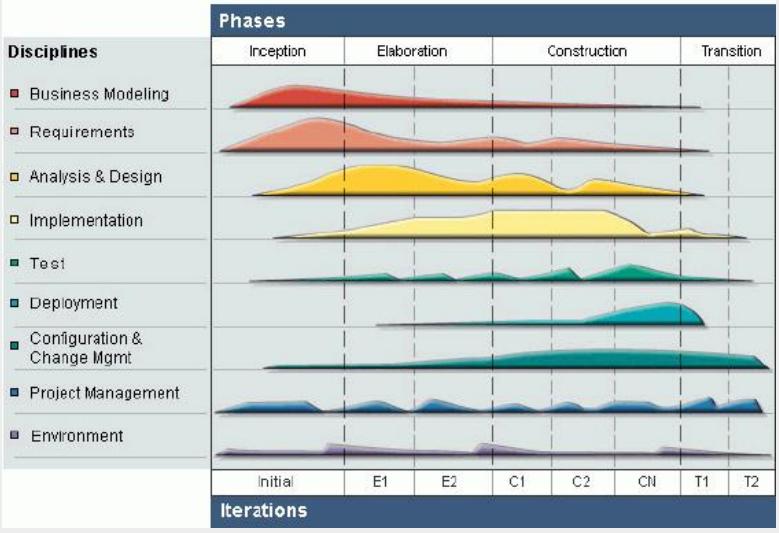
Construction

- The objective is to build the software system. The main focus is on the development of components of the system.

Transition

- The objective is to 'transit' the system from development into production. Activities include train the end users and beta testing the system.

RUP Phases and Disciplines



Nine Disciplines
The nine disciplines are performed iteratively throughout the four phases

1. Business Modeling– To understand the business of the organization, possibility of re-engineering business process is explored and potential strategies are evaluated.
2. Requirements - The scope of the project is defined and specification documents for functional and non-functional requirements are prepared.
3. Analysis and Design – The requirements are analyzed and architecture design of the system is made.
4. Implementation – The program source code is developed and unit testing is done.
5. Test - This discipline ensures quality of the system developed. It consists of finding bugs, ensuring that the system works as per the design of the system and meets all requirements mentioned in the specification documents.
6. Deployment - Includes planning and executing delivery of software and supporting documentations ready to be deployed and making the system available to end users.
7. Configuration and Change Management – This includes managing baselines of the project, accepting and managing change requirements, changing and delivering configuration items and managing releases.
8. Project management - This includes assigning tasks, managing risks, tracking progress etc. to ensure on time and within budget delivery of the product.
9. Environment – This includes ensuring proper tools are available whenever required.

Agile Development – Extreme Programming (XP)

XP is one of the most well-known agile development life cycle models.

Some characteristics of XP are:

- It promotes the generation of business stories to define the functionality
- It demands an on-site customer for continual feedback and to define and carry out functional acceptance testing
- It promotes pair programming and shared code ownership amongst the developers
- It states that component test scripts shall be written before the code is written and those tests are automated
- It states that integration and testing of the code shall happen several times a day
- It focuses on implementing the simplest solution to meet today's problems

Testing Phases

Unit (Component) testing

- Unit testing is code-based and performed primarily by developers to demonstrate that their smallest pieces of executable code function suitably.

Integration testing

- Integration testing demonstrates that two or more units or other integrations work together properly, and tends to focus on the interfaces specified in low-level design.

System testing

- System testing demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design.

Acceptance testing

- Acceptance testing is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.

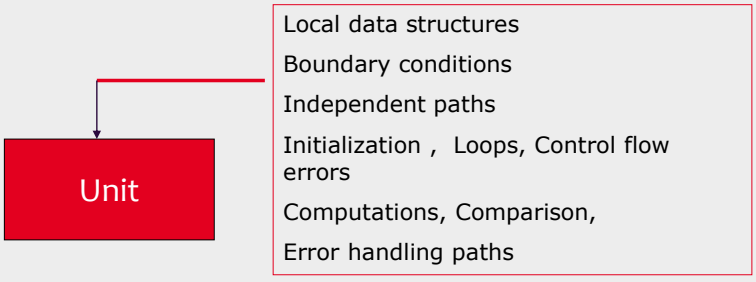
Introduction of Component Testing

- The most 'micro' scale of testing to test particular functions, procedures or code modules is called Component testing ; Also called as Module or Unit testing
- Typically done by the programmer and not by Test Engineers, as it requires detailed knowledge of the internal program design and code
- Purpose is to discover discrepancies between the unit's specification and its actual behavior
- Testing a form, a class or a stored procedure can be an example of unit testing

What is a Unit?
Synonyms are “component” and “module.”
The IEEE glossary says (for module):
A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.
A logically separable part of a program.

Component /Unit Testing

Unit testing uncovers errors in logic and function within the boundaries of a component.



The module interface is tested to ensure that information properly flows into and out of the program under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.

Test cases should uncover errors such as

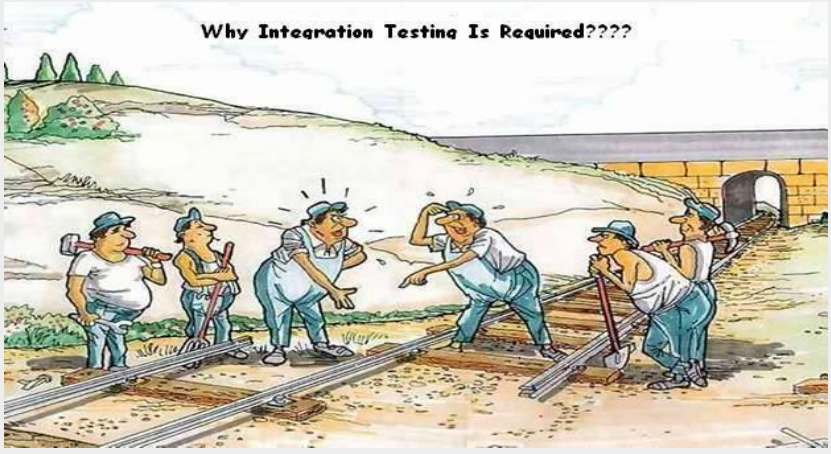
(1) comparison of different data types
(2) incorrect logical operators or precedence
(3) expectation of equality when precision error makes equality unlikely
(4) incorrect comparison of variables
(5) improper or nonexistent loop termination
(6) failure to exit when divergent iteration is encountered.

Introduction of Integration testing

- Testing of combined parts of an application to determine if they function together correctly
- The main three elements are interfaces, module combinations and global data structures
- Attempts to find discrepancies between program & its external specification (program's description from the point of view of the outside world)
- Testing a module to check if the component of the modules are integrated properly is example of integration testing

Interfaces are the means by which data is passed to and from modules. "Interface integrity" – to ensure that when data is passed to another module, by way of a call, none of the data becomes lost or corrupted. This loss or corruption can happen by number of ways- calling and receiving parameters may be of the wrong type and so the data appears in the receiving programs in a garbled form; there may be different number of calling and receiving parameters and so the data is lost; arrays may be of different lengths.
Global data structures are those pieces of data, maybe in the form of files, databases or variables, that are existing through out the entire system. Every module has access to global data structures and may alter the contents thereof. The effect of this again may create some unusual combination of data which reveals an error in another module.

Why Integration Testing is Required?



Types of Integration testing

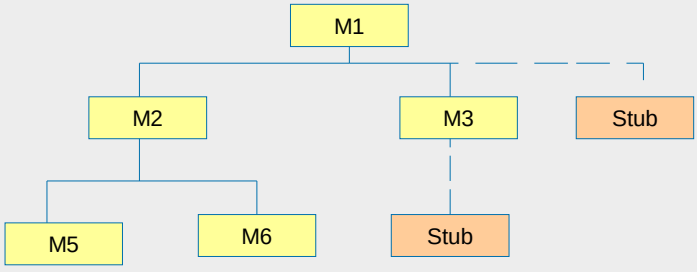
Modules are integrated by two ways.

- Non-incremental Testing (Big Bang Testing)
- Each Module is tested independently and at the end, all modules are combined to form a application
- Incremental Module Testing.
  - There are two types by which incremental module testing is achieved.
- Top down Approach
- Bottom up Approach

Top Down Integration Testing

Top Down Incremental Module Integration:

- Firstly top module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested.



Disadvantages:
Many tests are delayed until stubs are replaced by actual modules.
Time taken to develop stubs to perform the functions of the actual modules.

Advantage :
Fast

Top Down Integration Testing (Cont.)

Integration approach can be done Depth first or Breadth-first.

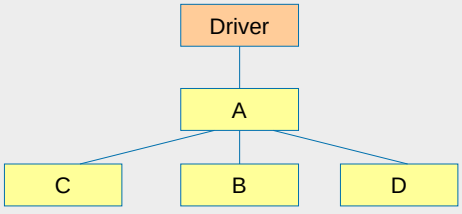
Top down testing

- The main control module is used as a test driver
- Stubs are substituted for all components directly subordinate to the main control module
- Depending on the approach subordinate stubs are replaced by actual components

Bottom Up Integration Testing

Bottom Up Incremental Module Integration:

- Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested. This continues till top most module is added to rest all and tested



Bottom Up Integration Testing (Cont.)

Bottom-Up testing

- Low-level components are combined into clusters (builds) that perform a specific sub function
- A driver is written to coordinate test case input and output
- Drivers are removed and clusters are combined moving upward in the program structure

Top Down vs. Bottom Up Testing

Top Down Testing	
Advantages	**Disadvantages**
Advantageous if major flaws occur toward the top of the program	Stub modules must be produced
Once the I/O functions are added, representation of test cases are easier	Stub Modules are often more complicated than they first appear to be.
Early skeletal Program allows demonstrations and boosts morale	Before the I/O functions are added, representation of test cases in stubs can be difficult
	Test conditions may be impossible, or very difficult, to create
	Observation of test output is more difficult
	Allows one to think that design and testing can be overlapped
	Induces one to defer completion of the testing of certain modules.

Top Down vs. Bottom Up Testing

Bottom Up testing	
Advantages	**Disadvantages**
Advantageous if major flaws occur toward the bottom of the program	Driver Modules must be produced
Test conditions are easier to create	The program as an entity does not exist until the last module is added
Observation of test results is easier	

Introduction to System Testing

- Test the software in the real environment in which it is to operate. (hardware, people, information, etc.)
- Observe how the system performs in its target environment, for example in terms of speed, with volumes of data, many users, all making multiple requests.
- Test how secure the system is and how can the system recover if some fault is encountered in the middle of procession.
- System Testing, by definition, is impossible if the project has not produced a written set of measurable objectives for its product.

Types of System Testing

- Functional Testing
- Performance Testing
- Volume Testing
- Load Testing
- Stress Testing
- Security Testing
- Web Security Testing
- Localization Testing
- Usability Testing
- Recovery Testing
- Documentation Testing
- Configuration Testing
- Installation Testing
- User Acceptance Testing
- Testing related to Changes : Re-Testing and Regression Testing
- Re-testing (Confirmation Testing)
- Regression Testing
- Exploratory Testing
- Maintenance Testing

Functional Testing

The main objective of functional testing is to verify that each function of the software application / system operates in accordance with the written requirement specifications

It is a black-box process

- Is not concerned about the actual code
- Focus is on validating features
- Uses external interfaces, including Application programming interfaces (APIs), Graphical user interfaces (GUIs) and Command line interfaces (CLIs)

Testing functionality can be done from two perspectives :

- Business-process-based testing uses knowledge of the business processes
- Requirements-based testing uses a specification of the functional requirements for the system as the basis for designing tests

Performance Testing

Performance

- Performance is the behavior of the system w.r.t. goals for time, space, cost and reliability

Performance objectives:

- Throughput : The number of tasks completed per unit time. Indicates how much work has been done within an interval
- Response time : The time elapsed during input arrival and output delivery
- Utilization : The percentage of time a component (CPU, Channel, storage, file server) is busy

Performance Testing (Cont.)

- The objective of performance testing is to devise test case that attempts to show that the program does not satisfy its performance objectives.
- To ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.
- To test response time and reliability by increased user traffic.
- To identify which components are responsible for performance degradation and what usage characteristics cause degradation to occur.

Volume Testing

This testing is subjecting the program to heavy volumes of data. For e.g.

- A compiler would be fed a large source program to compile
- An operating systems job queue would be filled to full capacity
- A file system would be fed with enough data to cause the program to switch from one volume to another.

Load Testing

Volume testing creates a real-life end user pressure for the target software. This tests how the software acts when numerous end users access it concurrently. For e.g.

- Downloading a sequence of huge files from the web
- Giving lots of work to a printer in a line

Stress Testing

Stress testing involves subjecting the program to heavy loads or stresses.

The idea is to try to “break” the system.

That is, we want to see what happens when the system is pushed beyond design limits.

It is not same as volume testing.

A heavy stress is a peak volume of data encounters over a short time.

In Stress testing a considerable load is generated as quickly as possible in order to stress the application and analyze the maximum limit of concurrent users the application can support.

Stress Testing(Cont.)

Stress tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

Example :

- Generate 5 interrupts when the average rate is 2 or 3
- Increase input data rate
- Test cases that require max. memory

Stress Tests should answer the following questions

- Does the system degrade gently or does the server shut down
- Are appropriate messages displayed ? E.g. Server not available
- Are transactions lost as capacity is exceeded
- Are certain functions discontinued as capacity reaches the 80 or 90 percent level

Security Testing

Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.

Security testing is the process of executing test cases that subvert the program's security checks.

Example :

- One tries to break the operating systems memory protection mechanisms
- One tries to subvert the DBMS's data security mechanisms
- The role of the developer is to make penetration cost more than the value of the information that will be obtained

Any computer based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities : hackers who attempt to penetrate systems for sport, disgruntled employees who attempt to penetrate for revenge, dishonest individuals who attempt to penetrate for illicit personal gain.

Firewalls – a filtering mechanism that is a combination of hardware and software that examines each incoming packet of information to ensure that it is coming from a legitimate source, blocking any data that are suspect.
Authentication – a verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.
Encryption – Protect sensitive data by modifying it in a way that makes it impossible to read by those with malicious intent.
Authorization – a filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes. (e.g. Userid , Passwords)

Web Security Testing

Web application security is a branch of Information Security that deals specifically with security of web applications.

It provides a strategic approach in identifying, analyzing and building a secure web applications.

It is performed by Web Application Security Assessment.

Why, Web Application Security?

1. Top breaches/fraud in recent times
2. Heartland Payment Systems - In January 2009 attackers were able to steal more than 130,000,000 credit card records
3. Virginia State Prescription Monitoring Program Records - Hackers stole 8.3 million records, erased the originals and created an encrypted backup of VPMP's database
4. Terrorists intercept US Drone unencrypted Video Feeds - Islamic terrorists have been able to hack into CIA state-of-the-art Predator drones with the help of just a 25.95 dollar off-the-shelf software, raising fears of remote control operated unmanned crafts being taken over and used against British and American targets.
5. Phishing attacks on banking sites – ICICI, SBI etc.,

Localization Testing

Localization translates the product UI and occasionally changes some settings to make it suitable for another region.

The test effort during localization testing focuses on

- Areas affected during localization, UI and content
- Culture/locale-specific, language specific and region specific areas

The goal of globalization testing is to detect potential problems in application design that could inhibit globalization. It makes sure that the code can handle all international support without breaking the functionality that would cause either data loss or display problems.

Automated testing is not an effective solution for localization/globalization testing because the default mouse click position changes with the language and also the text in the recordings don't obviously match for all languages.

Usability Testing

Usability is

- The effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment ISO 9241-11
- Effective–- Accomplishes user's goal
- Efficient-- Accomplishes the goal quickly
- Satisfaction–- User enjoys the experience

Test Categories and objectives

- Interactivity (Pull down menus, buttons)
- Layout
- Readability
- Aesthetics
- Display characteristics
- Time sensitivity
- Personalization

Usability testing can be formal, informal or heuristic based on the needs and the availability.

There are many frameworks formulated for usability testing like NIST has formulated a CIF (Common Industry Format) for reporting the findings of the test.

Interactivity – Are interaction mechanisms(pull down menus ,pointers) easy to understand

Layout – Are navigation mechanisms, content and functions placed in a manner that allows the user to find them quickly?

Readability – Is text well written and understandable ? Are graphic representation easy to understand?

Aesthetics – Do layout, color, typeface and related characteristics lead to ease of use? Do users feel comfortable with the look and feel.

Display Characteristics – optimal use of screen size and resolution.

Time sensitivity – Can important features, functions and content be used or acquired in a timely manner

Personalization – Does the web application tailor itself to the specific needs of different user categories or individual users?

Usability Testing (Cont.)

Using specialized Test Labs a rigorous testing process is conducted to get quantitative and qualitative data on the effectiveness of user interfaces

Representative or actual users are asked to perform several key tasks under close observation, both by live observers and through video recording

During and at the end of the session, users evaluate the product based on their experiences

Trainer notes

Recovery Testing

A system test that forces the software to fail in variety of ways, checks performed

- recovery is automatic (performed by the system itself)
- reinitialization
- check pointing mechanisms
- data recovery
- restarts are evaluated for correctness

This test confirms that the program recovers from expected or unexpected events. Events can include shortage of disk space, unexpected loss of communication

It Confirms that the program recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.

Documentation Testing

This testing is done to ensure the validity and usability of the documentation

This includes user Manuals, Help Screens, Installation and Release Notes

Purpose is to find out whether documentation matches the product and vice versa

Well-tested manual helps to train users and support staff faster

Generally, Technical Writers work to tight schedules, which often does not include documentation testing because there is no time. Besides, no one wants to take the risk of causing a rewrite or correcting product design and not shipping on schedule. Bad documentation has a ripple effect on the number of users it impacts such as Product Development, Training, and Customer Support.

Configuration Testing

Attempts to uncover errors that are specific to a particular client or server environment

Create a cross reference matrix defining all probable operating systems, browsers, hardware platforms and communication protocols

Test to uncover errors associated with each possible configuration

Configuration Test
Analyze system behaviour:
in various hardware and software configurations specified in the requirements
sometimes systems are built in various configurations for different users
for instance, a minimal system may serve a single user, other configurations for additional users.

Installation Testing

Installer is the first contact a user has with a new software!!!

Installation testing is required to ensure:

- Application is getting installed properly
- New program that is installed is working as desired
- Old programs are not hampered
- System stability is maintained
- System integrity is not compromised

User Acceptance Testing

A test executed by the end user(s) in an environment simulating the operational environment to the greatest possible extent, that should demonstrate that the developed system meets the functional and quality requirements

Not a responsibility of the Developing Organization

To test whether or not the right system has been created

Usually carried out by the end user

Two types are :

- User Acceptance Testing
- Operational Acceptance Test / Production Acceptance Test
- Contract Acceptance Testing
- Compliance acceptance testing / Regulation Acceptance Testing
- Alpha Testing
- Beta Testing

User Acceptance Testing (UAT) - Focuses mainly on the functionality thereby validating the fitness-for-use and is performed by the users and application managers

Operational Acceptance Test / Production Acceptance Test - Validates whether the system meets the requirements for operation and may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities

Contract Acceptance Testing - Is performed against a contract's acceptance criteria for producing custom-developed software

Compliance acceptance testing / Regulation Acceptance Testing - Is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

Alpha Testing – It is performed at the developer's site by a cross-section of potential users and members of the developer's organization

Beta Testing - It is performed by a cross-section of users who install it and use it under real-world working conditions

Testing related to Changes : Re-Testing and Regression Testing

The target of testing is the testing of changes.

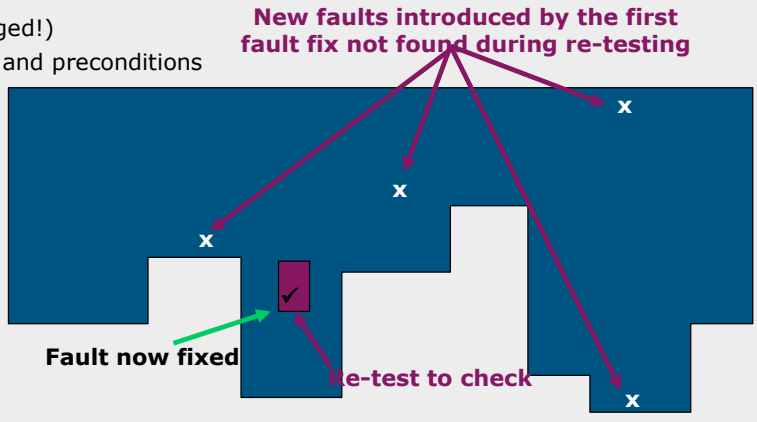
It includes below Test Types :

- Confirmation testing (Re-testing) – Re-execution of the test after defects are fixed. The test is executed in exactly the same way as it was the first time
- Regression testing – Ensures that the software is not adversely affected by the changes and critical functionality of the software is still intact. Generally there will be a regression test suite or regression test pack and executed whenever the software changes

Re-testing (Confirmation Testing)

Re-Running failed tests:

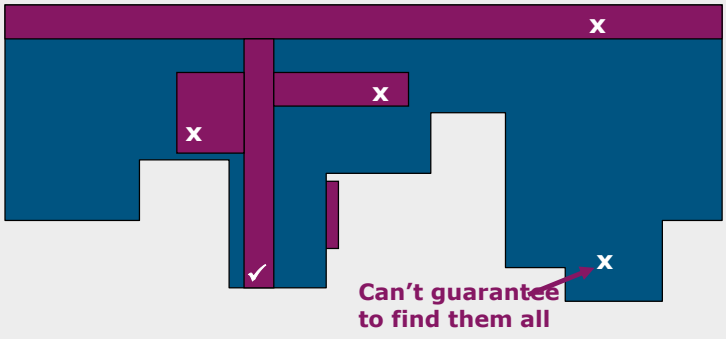
- Must be exactly repeatable
- Same environment, versions (except for the software which has been intentionally changed!)
- Same inputs and preconditions



Regression Testing

Regression tests are performed:

- To look for any unexpected side-effects
- After software changes, including faults fixed
- When the environment changes, even if application stays same
- For emergency fixes (possibly a subset)



Exploratory Testing

Also known as “Random” testing or “Ad-hoc” testing

Exploratory testing is simultaneous learning, test design, and test execution. (…James Bach)

A methodical approach-style is desirable

Among the hardest things to explain is something that everyone already knows. We all know how to listen, how to read, how to think, and how to tell anecdotes about the events in our lives. As adults, we do these things everyday. Yet the level of any of these skills, possessed by the average person, may not be adequate for certain special situations. Psychotherapists must be expert listeners and lawyers expert readers; research scientists must scour their thinking for errors and journalists report stories that transcend parlor anecdote.
So it is with exploratory testing (ET): simultaneous learning, test design and test execution.

Exploratory Testing - Tips

Test design Crafting
Careful Observation
Critical thinking
Diverse Ideas
Pooling resources (knowledge, learnings)

Test Design: An exploratory Test Engineer is first and foremost a test designer. Anyone can design a test accidentally, the excellent exploratory tester is able to craft tests that systematically explore the product. That requires skills such as the ability to analyze a product, evaluate risk, use tools, and think critically, among others.
Careful Observation: Excellent exploratory testers are more careful observers than novices, or for that matter, experienced scripted testers. The scripted tester need only observe what the script tells him to observe. The exploratory tester must watch for anything unusual or mysterious. Exploratory testers must also be careful to distinguish observation from inference, even under pressure, lest they allow preconceived assumptions to blind them to important tests or product behavior.
Critical Thinking: Excellent exploratory testers are able to review and explain their logic, looking for errors in their own thinking. This is especially important when reporting the status of a session of exploratory tests, or investigating a defect.
Diverse Ideas: Excellent exploratory testers produce more and better ideas than novices. They may make use of heuristics to accomplish this. Heuristics are mental devices such as guidelines, generic checklists, mnemonics, or rules of thumb.
Rich Resources: Excellent exploratory testers build a deep inventory of tools, information sources, test data, and friends to draw upon. While testing, they remain alert for opportunities to apply those resources to the testing at hand.

Maintenance Testing

Testing done after the system is deployed or on existing system is called Maintenance testing

It is different from maintainability testing, which defines how easy it is to maintain the system

It consists of two parts:

1. Testing the changes and defects
2. Regression tests

Impact and risk analysis is important activity performed to determine Test efforts

It is triggered by planned modifications, Ad-hoc corrective modifications, migration, or retirement of the system

Summary

In this lesson, you have learnt:

- Verification refers to a set of activities which ensures that software correctly implements a specific function.
- Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
- Different testing phases are
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing
- Exploratory testing is simultaneous learning, test design and test execution.

Review Question

Question 1: _____ is a Quality improvement process

Question 2: To test a function, the programmer has to write a _____, which calls the function to be tested and passes it test data

Question 3: Volume tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

- True/False

Question 4: Acceptance testing is not a responsibility of the Developing Organization

- True/False

Question 5: Difference between re-testing & regression testing is :

- re-testing is running a test again; regression testing looks for unexpected side effects
- re-testing looks for unexpected side effects; regression testing is repeating

Review Question: Match the Following

1. Beta testing	A. Volume testing
2. Response time	B. Exploratory testing
3. Aesthetics	C. Acceptance testing
	D. Documentation testing
	E. Performance testing
	F. Usability testing