

Software Testing Course Slide

Content :

Module - 1 Fundamental

Module - 2 Manual Testing

Module - 3 Context Testing

Module - 4 Bug Reporting Tools

Module - 5 Automation Tools

Module - 1 [Fundamentals]

Agenda

- Introduction to Software Engineering
- Software Development Life Cycle
- Software Engineering Models/Methodologies
- Waterfall Model/Methodologies
- Iterative & Incremental Model/Methodologies
- Spiral Model/Methodologies
- Agile Model/Methodologies
- Use Case Model/Methodologies
- Software Requirement Specification
- Object Oriented Programming
- Database and Structure Query Language

What is Software Engineering?

A **naive view:** Problem Specification Final Program But

...

- Where did the specification come from?
- How do you know the specification corresponds to the user's needs?
- How did you decide how to structure your program?
- How do you know the program actually meets the specification?
- How do you know your program will always work correctly?
- What do you do if the users' needs change?
- How do you divide tasks up if you have more than a one-person team?

What is Software Engineering?

Some Definitions and Issues

“Software engineering is the art of developing quality software on time and within budget.”

- Trade-off between perfection and physical constraints
- SE has to deal with real-world issues
- State of the art!
- Community decides on “best practice” + life-long duration

What is Software Engineering?

What is Software Engineering? (II)

“Software engineering is the multi-person construction of multi-version software” - Parnas

- Team-work
- Scale issue (“program well” is not enough)
+Communication Issue
- Successful software systems must evolve or perish
- Change is the norm, not the exception

What is Software Engineering?

What is Software Engineering? (III)

“software engineering is different from other engineering disciplines” —

- Summerville
- Not constrained by physical laws
- limit = human mind
- It is constrained by political forces
- balancing stake-holders

Software Development Life Cycle

- SDLC is a structure imposed on the development of a software product that defines the process for planning, implementation, testing, documentation, deployment, and ongoing maintenance and support. There are a number of different development models.

- A Software Development Life Cycle is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software.

- The methodology within the SDLC process can vary across industries and organizations, but standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition, and configuration of software systems.

SDLC Phases

Requirements Collection/Gathering	Establish Customer Needs
Analysis	Model And Specify the requirements- "What"
Design	Model And Specify a Solution – "Why"
Implementation	Construct a Solution In Software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to the new requirements

Requirement Gathering

- Features
- Usage scenarios
- Although requirements may be documented in written form, they may be incomplete, unambiguous, or even incorrect.
- Requirements will Change!
 - Inadequately captured or expressed in the first place
 - User and business needs change during the project
 - Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.
 - Build constant feedback into the project plan
 - Plan for change
 - Early prototyping [e.g., UI] can help clarify the requirements
 - Functional and Non-Functional

Requirement Gathering(Cont...)

- Requirements definitions usually consist of **natural language**, supplemented by (e.g., UML) **diagrams and tables**.
- Three types of problems can arise:
 - **Lack of clarity:** It is hard to write documents that are both **precise and easy-to-read**.
 - **Requirements confusion:** **Functional and Non-functional** requirements tend to be intertwined.
 - **Requirements Amalgamation:** Several **different requirements** may be expressed together.

Requirement Gathering(Cont...)

- Types of Requirements:

- **Functional Requirements:** describe system **services or functions.**
 - Compute sales tax on a purchase
 - Update the database on the server
- **Non-Functional Requirements:** are **constraints** on the system or the development process.
- **Non-functional requirements may be more critical than functional requirements.**
- **If these are not met, the system is useless!**

Analysis Phase

- The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.
- This phase defines the problem that the customer is trying to solve.
- The deliverable result at the end of this phase is a requirement document.
- Ideally, this document states in a clear and precise fashion what is to be built.
- This analysis represents the “**what**” phase.
- The requirement documentaries to capture the requirements from the customer's perspective by defining goals.

Analysis Phase

- This phase starts with the requirement document delivered by the requirement phase and maps the requirements into architecture.
- The architecture defines the components, their interfaces and behaviors.
- The deliverable design document is the architecture.
- This phase represents the “**how**” phase.
- Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.
- The design may include the usage of existing components.

Design Phase

- Design Architecture Document
- Implementation Plan
- Critical Priority Analysis
- Performance Analysis
- Test Plan
- The Design team can now expand upon the information established in the requirement document.
- The requirement document must guide this decision process.
- Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product.
- The architecture team also converts the typical scenarios into a test plan.

Implementation Phase

- In the implementation phase, the team builds the components either from scratch or by composition.
- Given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.
- For example, a component may be narrowly designed for this particular system, or the component may be made more general to satisfy a reusability guideline.
 - Implementation - Code
 - Critical Error Removal
- The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.
- The end deliverable is the product itself. There are already many established techniques associated with implementation.

Testing Phase

- Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.
- It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.
- A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version.
- Quality is a distinguishing attribute of a system indicating the degree of excellence.
- Regression Testing
- Internal Testing
- Unit Testing
- Application Testing
- Stress Testing

Testing Phase(Cont...)

- The testing phase is a separate phase which is performed by a different team after the implementation is completed.
- There is merit in this approach; it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times.
- Unfortunately, delegating (alternate) testing to another team leads to as lack (dull) attitude regarding quality by the implementation team.
- If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases.
- an attitude change must take place to guarantee quality. Regardless if testing is done after the-fact or continuously, testing is usually based on a regression technique split into several major focuses, namely internal, unit, application, and stress.

Maintenance Phase

- Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.
- Software maintenance is also one of the phases in the System Development Life Cycle (SDLC), as it applies to software development. The maintenance phase is the phase which comes after deployment of the software into the field.
- The developing organization or team will have some mechanism to document and track defects and deficiencies.
- configuration and version management
- reengineering (redesigning and refactoring)
- updating all analysis, design and user documentation
- Repeatable, automated tests enable evolution and refactoring

Maintenance Phase(Cont...)

Maintenance is the process of changing a system after it has been deployed.

- **Corrective maintenance:** identifying and repairing defects
- **Adaptive maintenance:** adapting the existing solution to the **new platforms**.
- **Perfective Maintenance:** implementing the **new requirements**

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “**maintenance**”!

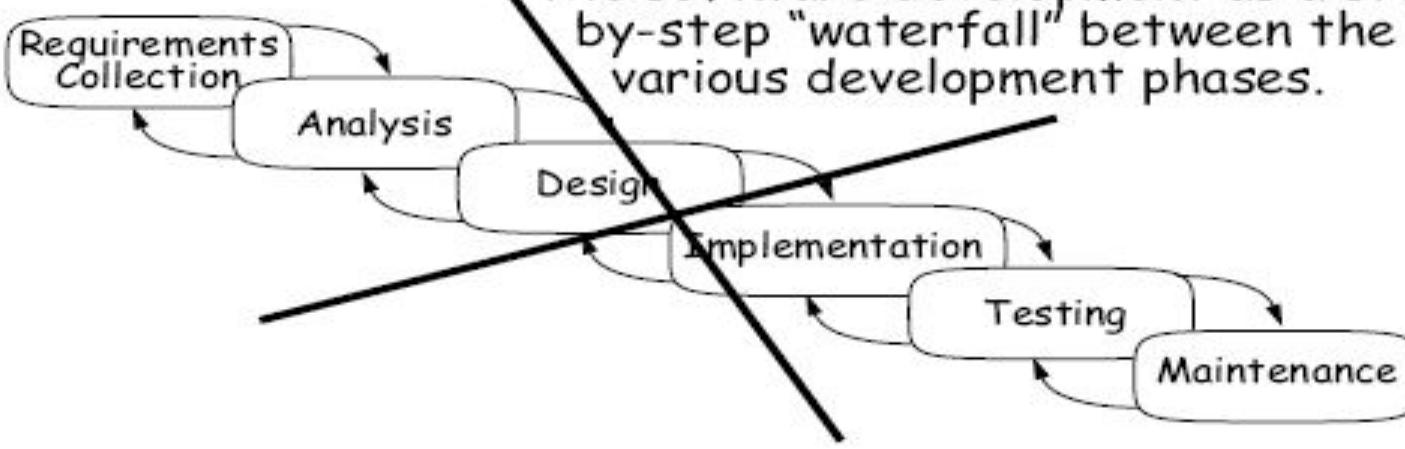
- Software just like most other products is typically released with a known set of defects and deficiencies.
- The software is released with the issues because the development organization decides the utility and value of the software at a particular level of quality outweighs the impact of the known defects and deficiencies.

Software Testing Methodologies

- Introduction
- Waterfall Model/Methodology (Classical Software Cycle)
- Iterative & Incremental Model/Methodology
- Spiral Model/Methodology
- Agile Model/Methodology
- Use Case

Waterfall Model (Classical Software Cycle)

The classical software lifecycle models the software development as a step-by-step "waterfall" between the various development phases.



The waterfall is unrealistic for many reasons, especially:

- Requirements must be "**frozen**" too early in the life cycle
- Requirements are **validated too late**

Applications(When to use?)

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

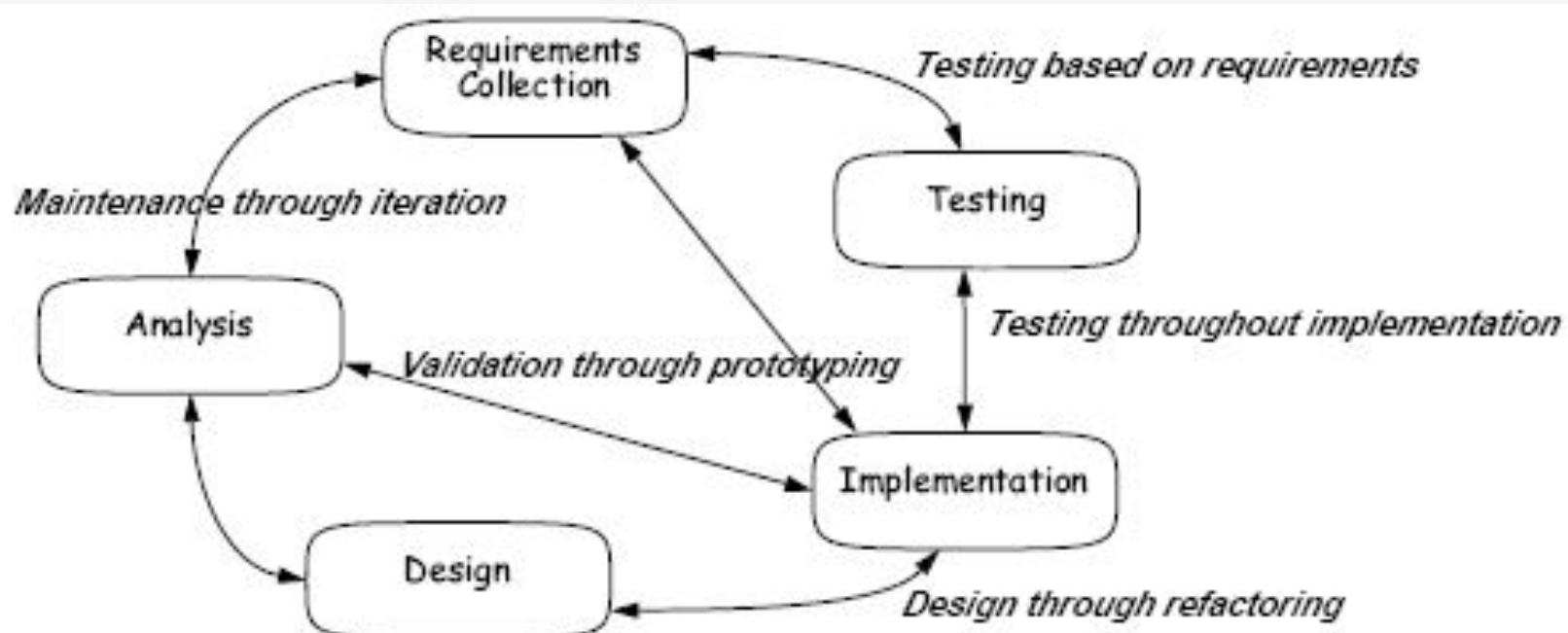
Pros (Why Waterfall Model)

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Cons (Why not Waterfall Model)

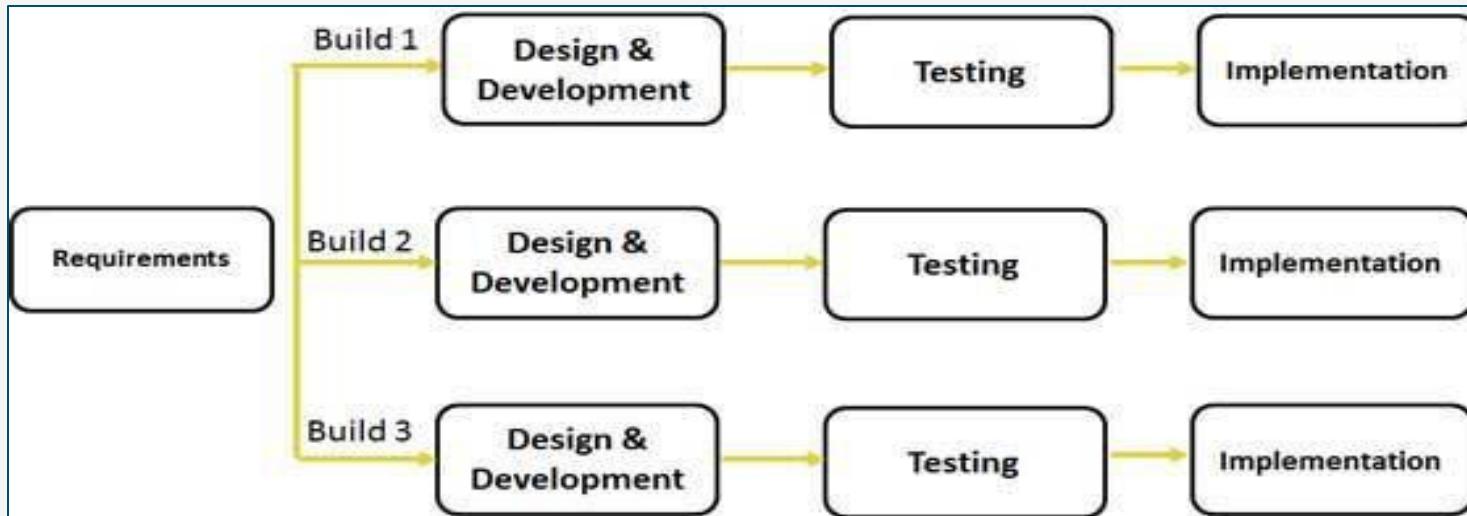
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Model/Methodology



- In Practice, development is always iterative, and all activates progress in parallel.
- If the waterfall model is pure fiction, why is it still the standard software process?

Iterative & Incremental Model



Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through **repeated cycles (iterative)** and in smaller portions at a time (**incremental**).

Applications(When to use?)

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

Pros

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.

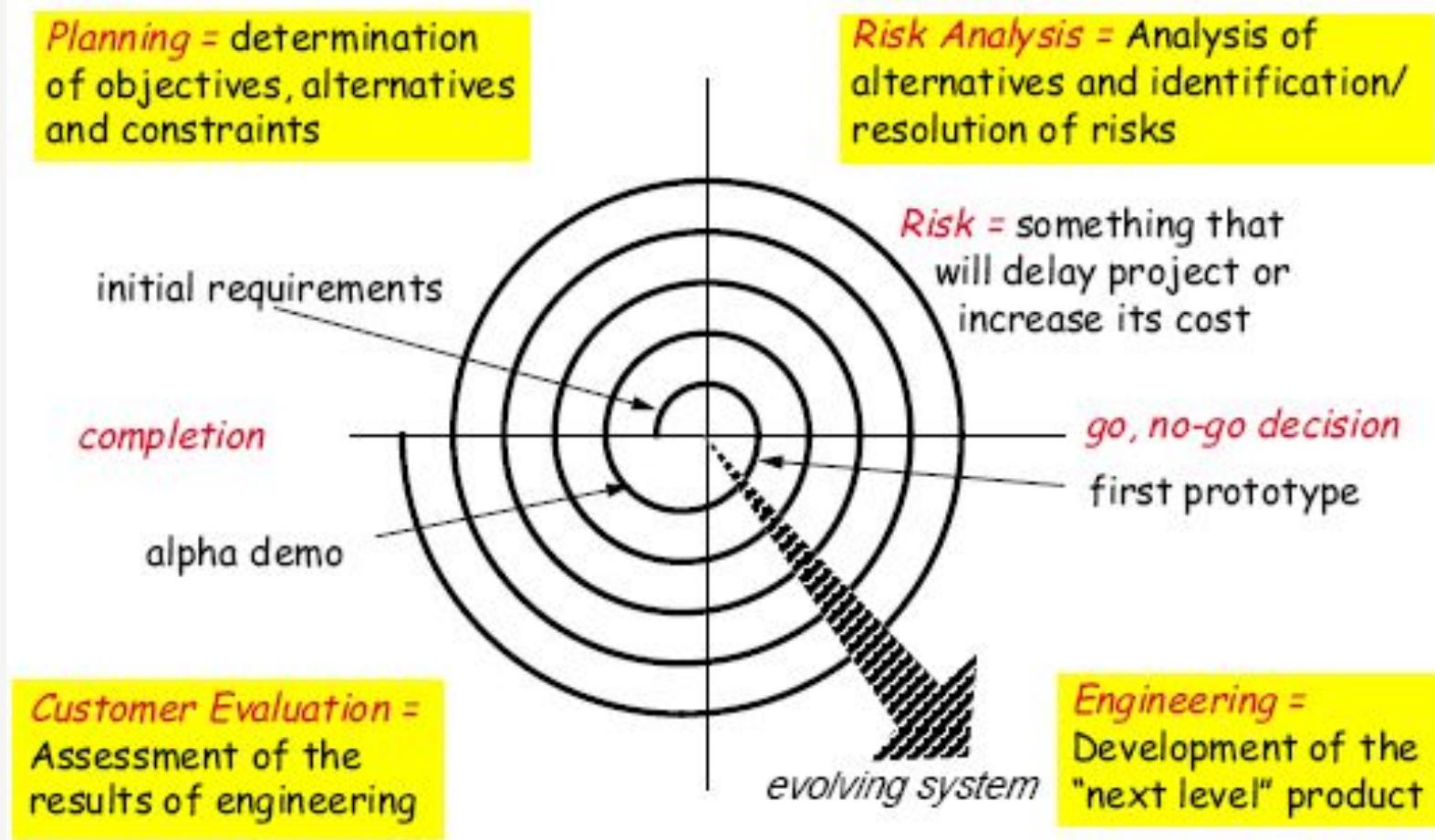
Pros

- Easier to manage risk - High risk part is done first.
- With every increment operational product is delivered.
- Issues, challenges & risks identified from each increment can be utilized/applied to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- During life cycle software is produced early which facilitates customer evaluation and feedback.

Cons

- More resources may be required.
- Although cost of change is lesser but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which a risk is.
- Highly skilled resources are required for risk analysis.
- Project's progress is highly dependent upon the risk analysis phase.

Bohem's Spiral Model



Application

Spiral Model is very widely used in the software industry as it is in sync with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the customer as well as the development firms. Following are the typical uses of Spiral model:

- When costs there are a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which are usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Pros (Why It works)

- Changing requirements can be accommodated.
- Allows for extensive use of prototypes
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

Cons (Why It doesn't work)

- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Agile Model/Methodology

- Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds.
- These builds are provided in iterations.
- Each iteration typically lasts from about one to three weeks.
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- At the end of the iteration a working product is displayed to the customer and important stakeholders.

What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

- Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

Pros

- Is a very realistic approach to software development
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required
- Easy to manage
- Gives flexibility to developers

Cons

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Use-case

A use-case **is the specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.**

- Ex: buy a DVD through the internet

A scenario **is a particular trace of action occurrences, starting from a known initial state**

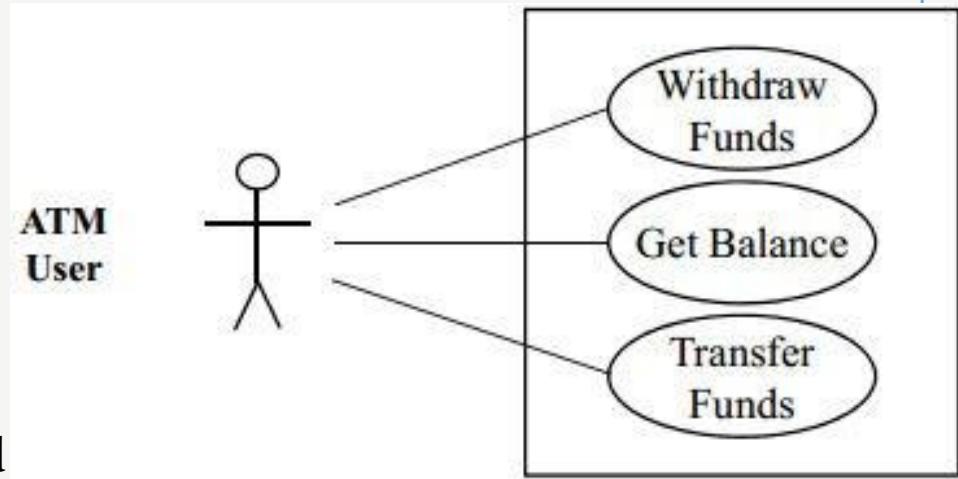
- Ex: connect to my DVD.com
- Ex: go to “search” page.

Use cases are deceptively simple tools for describing the behavior of software or systems.

A use case contains a textual description of all of the ways which the intended users could work with the software or system.

Example of ATM

- Use cases are commonly elaborated (or documented)
- Elaboration is first written textually
 - Details of operation
 - Alternatives model choices and conditions during execution
- **Actors:** Humans or software components that use the software being modeled
- **Use cases:** Shown as circles or ovals
- **Node Coverage:** Try each use case once ...
- **Use Case graphs, by themselves, are not useful for testing**



Software Requirement Specification

- A software requirements specification (SRS) is a complete description of the behavior of the system to be developed.
- It includes a set of use cases that describe all of the interactions that the users will have with the software.
- Use cases are also known as functional requirements. In addition to use cases, the SRS also contains nonfunctional (or supplementary) requirements.
- Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).
- Recommended approaches for the specification of software requirements are described by **IEEE 830-1998**.
- This standard describes possible structures, desirable contents, and qualities of a software requirements specification.

Types of Requirements

Requirements are categorized in several ways. The following are common categorizations of requirements that relate to technical management:

- Customer Requirements
- Functional Requirements
- Non-Functional Requirements

Customer Requirements

The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing:

- Operational distribution or deployment: Where will the system be used?
- Mission profile or scenario: How will the system accomplish its mission objective?
- Performance and related parameters: What are the critical system parameters to accomplish the mission?
- Utilization environments: How are the various system components to be used?
- Effectiveness requirements: How effective or efficient must the system be in performing its mission?
- Operational life cycle: How long will the system be in use by the user?
- Environment: What environments will the system be expected to operate in an effective manner?

Functional Requirements

Functional Requirements are very important system requirements in the system design process. These requirements are the technical specifications, system design parameters and guidelines, data manipulation, data processing, and calculation modules etc, of the proposed system.

For Example: The following are the requirements of Google Email Service

- The system shall support the ability to receive emails
- The system shall support the ability to send emails
- The system shall support the ability to create new folders
- The system shall support the ability to filter emails in different folders
- The system shall support the ability to attach different kind of attachments
- The system shall support the ability to create and maintain address book
- The system shall support the ability to create unlimited user accounts with different email addresses

Non-Functional Requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors. Non-functional requirements are qualities or standards that the system under development must have or comply with, but which are not tasks that will be automated by the system.

Example non-functional requirements for a system include:

- system must be built for a total installed cost of \$1,050,000.00
- system must run on Windows Server 2003
- system must be secured against Trojan attacks

A software development methodology helps to identify, document, and realize the requirements. Non functional requirements can be divided into following categories:

1. Usability
2. Reliability
3. Performance
4. Security

Product & Project Based Application

Before we delve into the differences, for a better clarity, I would like to explain what is a software product and a software project.

Software product: A software application that is **developed by a company with its own budget**. The requirements are driven by market surveys. The developed product is then sold to different customers with licenses.

Example for software products: Tally (by TCS), Acrobat reader/writer (Adobe), Internet Explorer (MS), Finale (Infosys), Windows (MS), QTP (HP) etc.

Software project: A software application that is **developed by a company with the budget from a customer**. In fact, the customer gives order to develop some software that helps him in his business. Here, the requirements come from the customer.

Example for software projects: A separate application ordered by a manufacturing company to maintain its office inventory etc.

Object Oriented Programming



- Programming is like writing.
- If you can write a demonstration, you can make a program.
- So, programming is also easy.
- But, actually, programming is not so easy, because a real good program is not easily programmed. It needs the programmers' lots of wisdom, lots of knowledge about programming and lots of experience.
- It is like writing, to be a good writer needs lots of experience and lots of knowledge about the world.
- Learning and practise is necessary

Object-Oriented Languages

An object-based programming language is one which easily supports object-orientation.

Smalltalk : 1972-1980

- Founder of Alan Kay

C++ : 1986, Bjarne Stroustrup

Java(Oak) : 1992 (Smalltalk + C++)

- Founder of James Gosling

- Developed by Sun Microsystem overtake by Oracle.

C# :

- Developed at Microsoft by Anders Hejlsberg et al, 2000

- Event driven, object oriented, visual programming language (C++ and Java)

Others:

- Effile, Objective-C, Ada, ...

What is OOP?



- Identifying **objects** and assigning **responsibilities** to these objects.
- Objects communicate to other objects by sending **messages**.
- Messages are received by the **methods** of an object
- **An object is like a black box.**
- **The internal details are hidden.**
- Object is derived from abstract data type
- Object-oriented programming has a web of interacting objects, each house-keeping its own state.
- Objects of a program interact by sending messages to each other.

Everything in the world is an object

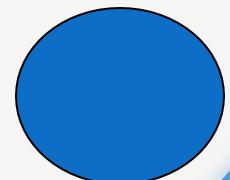
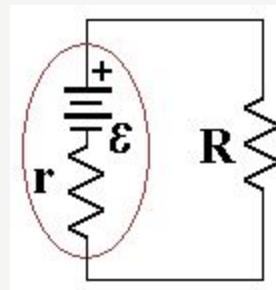
- A flower, a tree, an animal
- A student, a professor
- A desk, a chair, a classroom, a building
- A university, a city, a country
- The world, the universe
- A subject such as CS, IS, Math, History, ...

Concepts of OO

- Object
- Class
- Encapsulation
- Inheritance
- Polymorphism
 - Overriding
 - Overloading
- Abstraction

What is an object?

- Tangible Things as a car, printer, ...
- Roles as employee, boss, ...
- Incidents as flight, overflow, ...
- Interactions as contract, sale, ...
- Specifications as colour, shape, ...



So, what are objects?

An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.

An "object" is anything to which a concept applies.

This is the basic unit of object oriented programming(OOP).

That is both data and function that operate on data are bundled as a unit called as object.



The two parts of an object

Object = Data + Methods

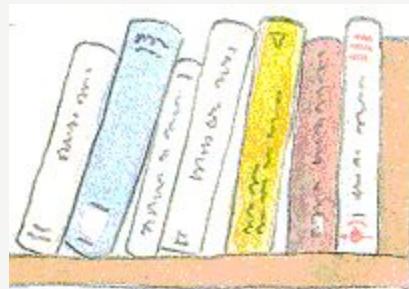
or

to say the same differently

An object has the responsibility to *know* and the responsibility to *do*.



=



+



Class

- When you define a class, you define a **blueprint for an object**.
- This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
- A class represents an **abstraction of the object and abstracts the properties and behavior of that object**.
- Class can be considered as the blueprint or definition or a template for an object and describes the properties and behavior of that object, but without any actual existence.
- **An object is a particular instance of a class** which has actual existence and there can be many objects (or instances) for a class.
- In the case of a car or laptop, there will be a blueprint or design created first and then the actual car or laptop will be built based on that.
- We do not actually buy these blueprints but the actual objects.

The two steps of Object Oriented Programming

- **Making Classes:** Creating, extending or reusing abstract data types.
- **Making Objects interact:** Creating objects from abstract data types and defining their relationships.

Encapsulation

Encapsulation is the practice of including in an object everything it needs hidden from other objects. The internal state is usually not accessible by other objects.

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

Encapsulation in Java is the process of wrapping up of data (properties) and behavior (methods) of an object into a single unit; and the unit here is a Class (or interface).

Encapsulate in plain English means *to enclose or be enclosed in or as if in a capsule*. In Java, a class is the capsule (or unit).

Encapsulation(Cont...)

- In Java, everything is enclosed within a class or interface, unlike languages such as C and C++, where we can have global variables outside classes.

- Encapsulation enables **data hiding**, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user.

- We can expose our operations hiding the details of what is needed to perform that operation.

- We can protect the internal state of an object by hiding its attributes from the outside world (*by making it private*), and then exposing them through setter and getter methods. Now modifications to the object internals are only controlled through these methods.

Abstraction

Abstraction is the representation of the essential features of an object. These are ‘encapsulated’ into an *abstract data type*.

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

For example, a database system hides certain details of how data is stored and created and maintained.

Similar way, C++ classes provides different methods to the outside world without giving internal detail about those methods and data.

In plain English, abstract means a concept or idea not associated with any specific instance and does not have a concrete existence.

Abstraction(Cont...)

- Abstraction in Object Oriented Programming refers to the ability to make a class abstract.
- Abstraction captures only those details about an object that are relevant to the current perspective.
- Abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. Java provides interfaces and abstract classes for describing abstract types.
- An **interface** is a contract or specification without any implementation. An interface can't have behavior or state.
- An **abstract class** is a class that cannot be instantiated. All other functionality of the class still exists. Abstract classes can have state and can be used to provide a skeletal implementation.
- A detailed comparison of interfaces and abstract classes can be found at **interface vs abstract-class**.

Polymorphism

- **Polymorphism means “having many forms”.**
- **It allows different objects to respond to the same message in different ways, the response specific to the type of the object.**
- The most important aspect of an object is its **behaviour** (the things it can do). A behaviour is initiated by **sending a message** to the object (usually by calling a method).
- The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism.



Polymorphism

- Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.
- E.g. the message *displayDetails()* of the Person class should give different results when send to a Student object (e.g. the enrolment number).
- **The ability to change form is known as polymorphism.**
- There are two types of polymorphism in Java
 - Compile time polymorphism(Overloading)
 - Runtime polymorphism(Overriding)



Overloading

- The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

- The **same method name (method overloading) or operator symbol (operator overloading)** can be used in different contexts.

- In method overloading, **multiple methods having same name can appear in a class, but with different signature.**

- And based on the number and type of arguments we provide while calling the method, the correct method will be called.

- Java doesn't allow operator overloading yet + is overloaded for class String. The '+' operator can be used for addition as well as string concatenation.

Inheritance

Inheritance means that one class inherits the characteristics of another class. This is also called a “is a” relationship

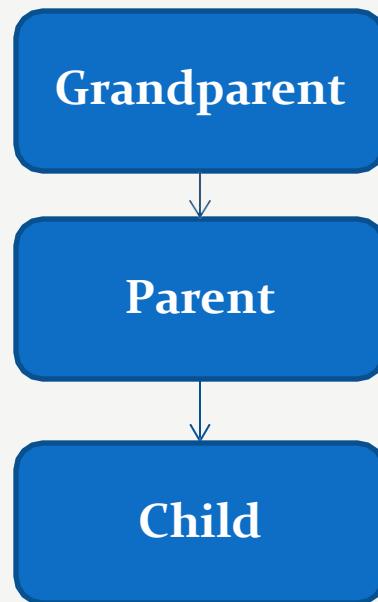
One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

Inheritance describes the relationship between two classes. A class can get some of its characteristics from a parent class and then add unique features of its own.

Inheritance

- In general, Java supports single-parent, multiple-children inheritance and multilevel inheritance (Grandparent-> Parent -> Child) for classes and interfaces. Java supports multiple inheritances (multiple parents, single child) only through interfaces.
- In a class context, inheritance is referred to as implementation inheritance, and in an interface context, it is also referred to as interface inheritance.



Inheritance(Cont...)

For example consider a Vehicle parent class and its child class Car.

- Vehicle class will have all common properties and functionalities for all vehicles in common and Car will inherit those common properties from the Vehicle class and then add those properties which are specific to a car.
- Here, Vehicle is known as base class, parent class, or super class.
- Car is known as derived class, Child class or subclass.

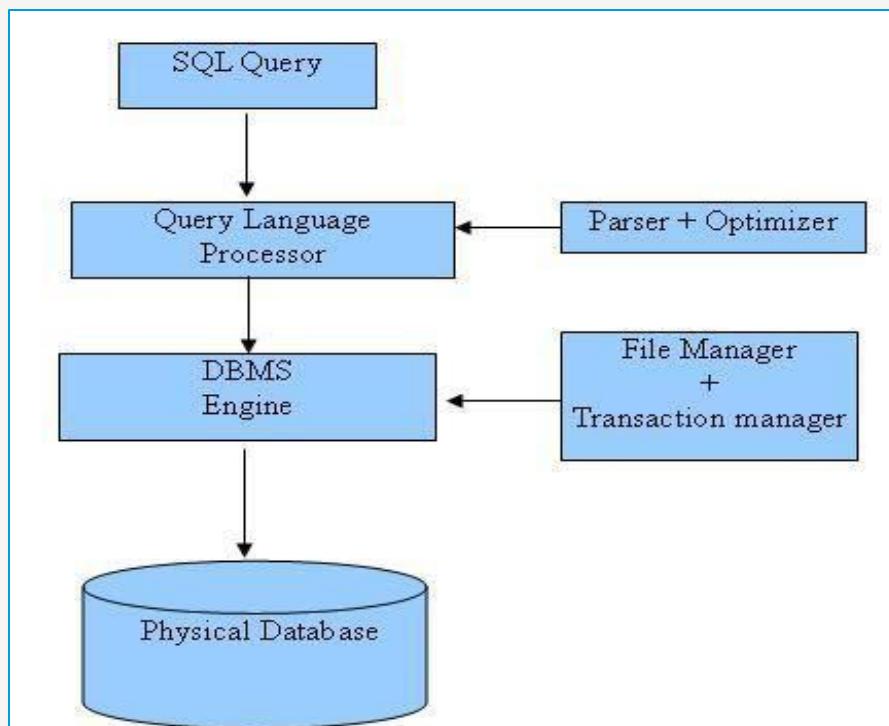
A car *is a* vehicle

A dog *is an* animal

A teacher *is a* person

DataBase and SQL Objectives

“The large majority of today's business applications revolve around relational databases and the SQL programming language (Structured Query Language). Few businesses could function without these technologies...”



Relational Databases

RDBMS stands for **Relational Database Management System**. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

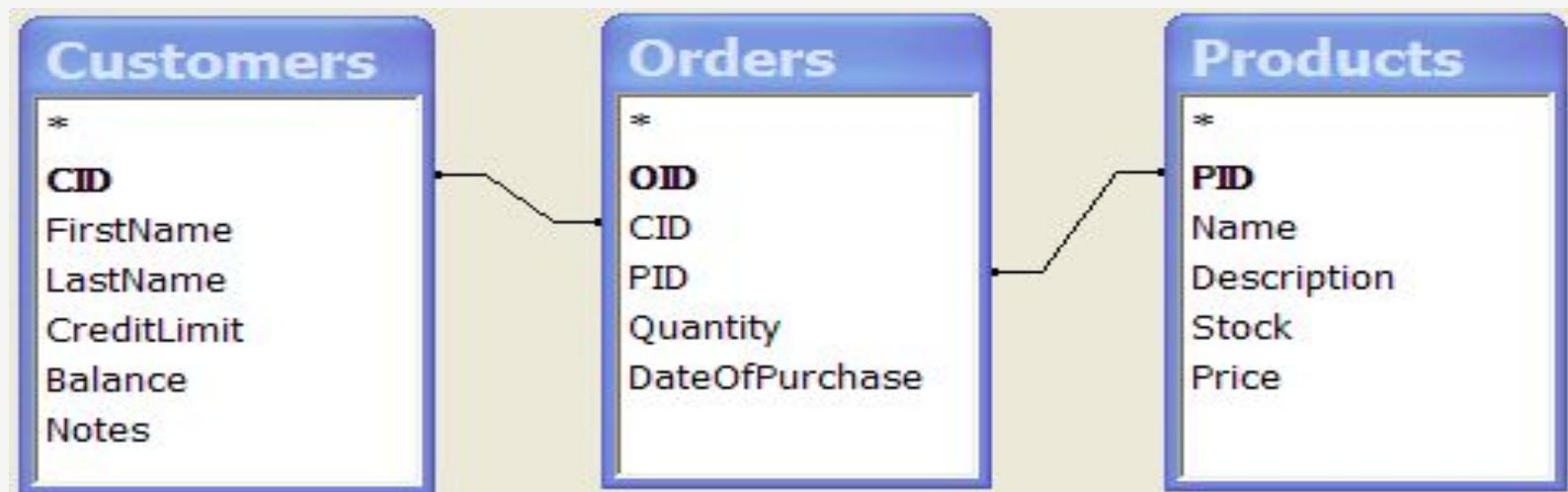
A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by **E. F. Codd**.

Most of today's databases are relational:

- database contains 1 or more *tables*
 - table contains 1 or more *records*
 - record contains 1 or more *fields*
 - fields contain the data
- So why is it called "relational"?
- tables are related (*joined*) based on common fields

Example

- Here's a simple **database schema** for tracking sales
 - 3 tables, related by primary keys (CID, OID, PID)
 - primary keys (in **boldface**) are unique record identifiers
 - customer may place order for one product at a time...



Customers...

- Here's some data for the Customers table
 - ignore last row, it's a MS Access mechanism for adding rows...

	CID	FirstName	LastName	CreditLimit	Balance	Notes
▶	1	Jim	Bag	\$1,000.00	\$0.00	works at the gym
	3	Kathie	O'Dahl	\$9,999.99	\$0.00	a friend with a special name!
	5	Bryan	Lore	\$1,000.00	\$900.00	a brother-in-law
	6	Amy	Lore	\$1,000.00	\$100.00	a sister-in-law
	14	Bill	Gates	\$2,000,000,000.00	\$89,992.00	
	116	Jane	Doe	\$1,000.00	\$420.00	
	666	Bad	Guy	\$1,000,000.00	\$235,000.00	a very bad guy...
*	0			\$0.00	\$0.00	

Products...

- Here's some data for the Products table

	PID	Name	Description	Stock	Price
▶	1	Flying Squirrels	yes, they really do fly!	3	\$899.99
	2	Cats		100	\$19.99
	3	Dogs	we only carry dalmations	20	\$79.03
	4	Ants		50000	\$0.09
	5	Birds		1000	\$4.95
	6	Elephants		10	\$389.95
	7	Red Fire Ants		10000	\$0.49
	8	Racoons		25	\$2.25
*	0			0	\$0.00

Record:  1  of 8

Orders...

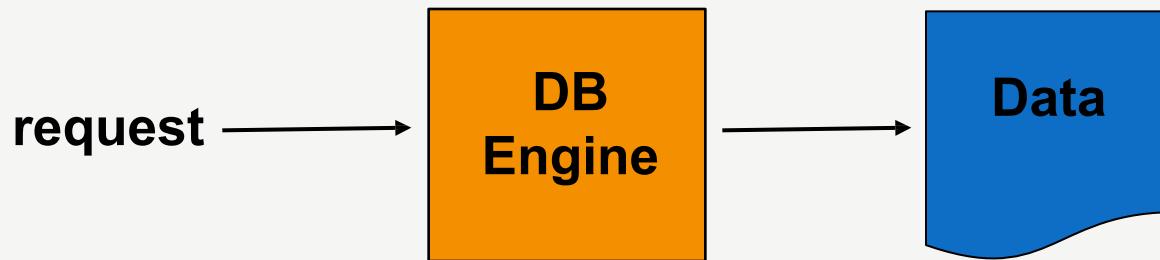
- Here's some data for the Orders table
 - how do you read this?
 - e.g. order #9906 states Kathie O'Dahl purchased 600 Ants
 - must join tables together to figure that out...

	OID	CID	PID	Quantity	DateOfPurchase
▶	9906	3	4	600	Wednesday, June 28, 2000
	12351	116	4	100	Tuesday, January 01, 2002
	22209	1	2	2	Thursday, January 03, 2002
	22210	1	2	1	Friday, June 28, 2002
	33410	1	3	1	Tuesday, October 01, 2002
*	0	0	0	0	

Database Management Systems

- A DBMS consists of 2 main pieces:

- the data
- the DB engine
- the data is typically stored in one or more files

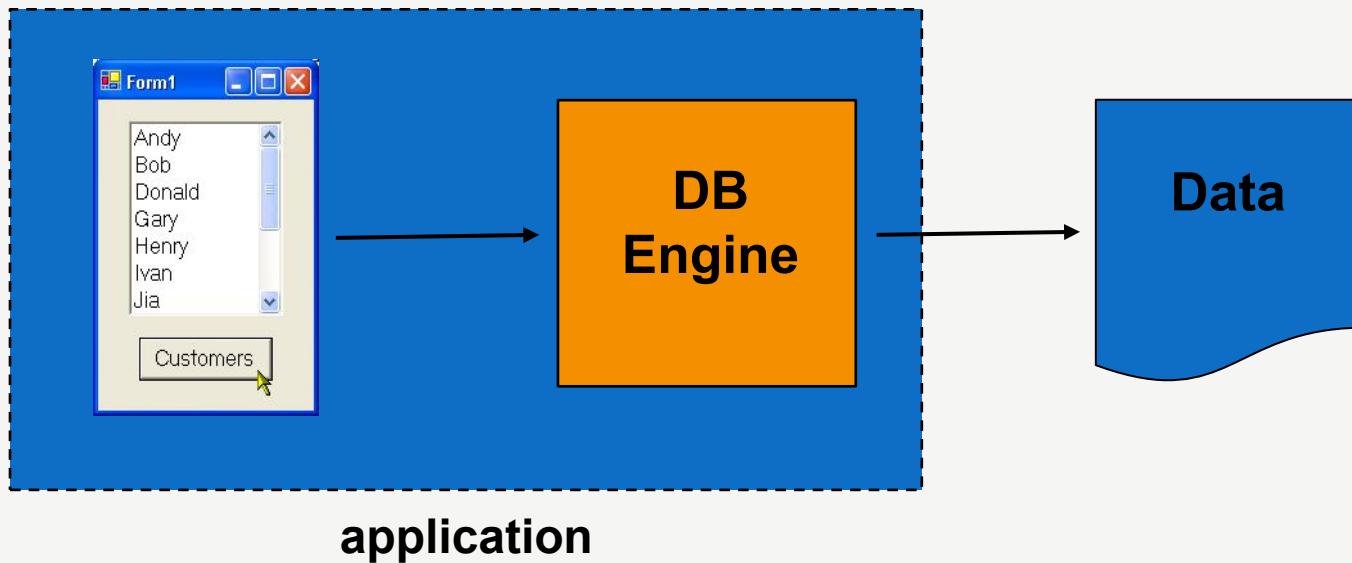


- Two most common types of DBMS are:

- Local
- Server

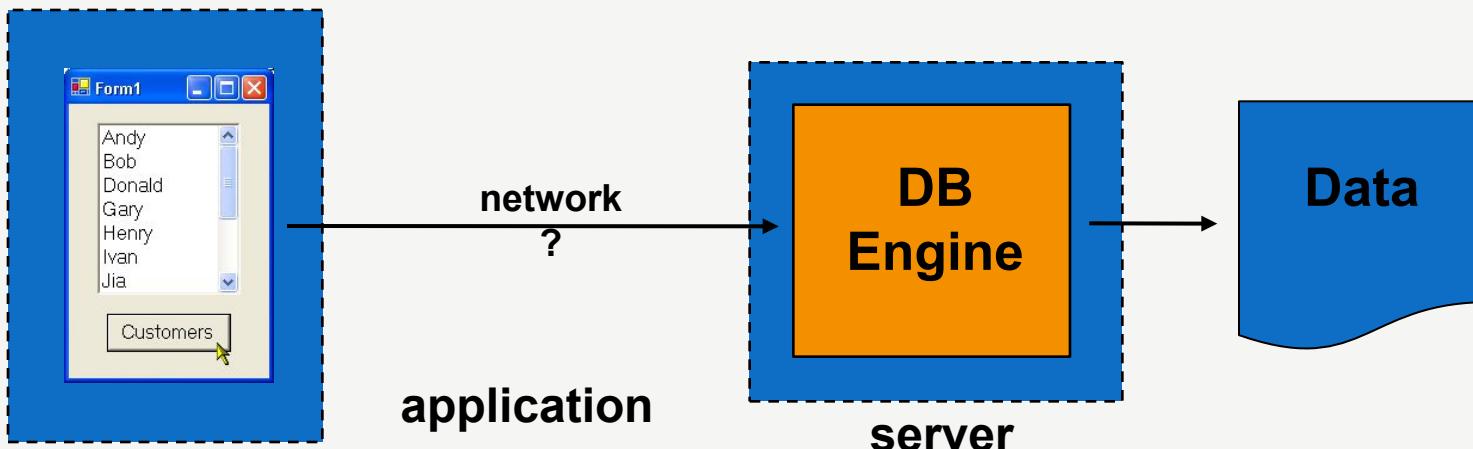
Local DBMS

- A local DBMS is where DB engine runs as part of application
- Example?
- MS Access
- underlying DB engine is JET ("Joint Engine Technology")



Server DBMS

- A server DBMS is where DB engine runs as a separate process
 - typically on a different machine (i.e. server)
- Examples?
 - MS SQL Server, Oracle, DB2, MySQL



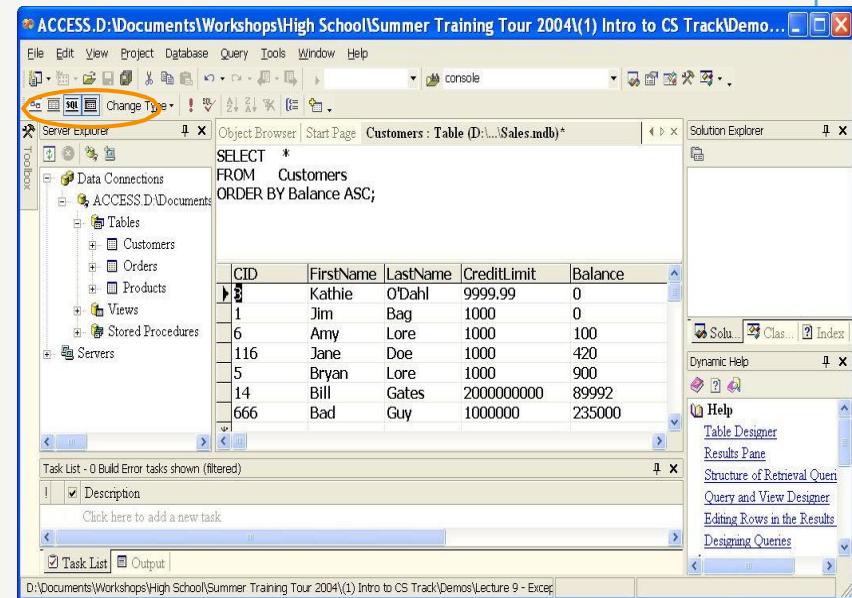
Databases & Visual Studio .NET

Most databases ship with tools for opening / manipulating DB

- MS Access database: use the MS Access Office product
- SQL Server database: use Query Analyzer

But you can also use Visual Studio .NET!

- View menu, Server Explorer
- right-click on Data Connections
- Add Connection...
- MS Access database:
 - Provider: JET 4.0 OLE DB
 - Connection:browse...
 - Connection:test...
- Drill-down to Tables
- Double-click on a table
- "Show SQL Pane" via toolbar



Structure Query Language

SQL tutorial gives unique learning on **Structured Query Language** and it helps to make practice on SQL commands which provides immediate results.

SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.

SQL is an **ANSI (American National Standards Institute)** standard but there are many different versions of the SQL language.

SQL is the standard programming language of relational DBs

SQL is a standard computer language for accessing and manipulating databases.

SQL is a great example of a **declarative** programming language

- your declare what you want, DB engine figures out how...

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL, ANSI SQL
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

What is SQL? (Cont...)

- SQL stands for Structured Query Language
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn
- SQL is written in the form of *queries*
- *action queries* insert, update & delete data
- *select queries* retrieve data from DB

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process.

- These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc.
- Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

SQL Commands

- **DDL** – Data Definition Language
- **DML** – Data Manipulation Language
- **DCL** – Data Control Language
- **DQL** – Data Query Language

SQL Join Types

- **INNER JOIN**: returns rows when there is a match in both tables.
- **LEFT JOIN**: returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN**: returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN**: returns rows when there is a match in one of the tables.

DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DQL – Data Query Language

Command	Description
SELECT	Retrieves certain records from one or more tables

DML – Data Manipulation Language

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL – Data Control Language

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

Create, Drop, Use Database Syntax

- **SQL CREATE DATABASE STATEMENT**

```
CREATE DATABASE database_name;
```

- **SQL DROP DATABASE Statement:**

```
DROP DATABASE database_name;
```

- **SQL USE STATEMENT**

```
USE DATABASE database_name;
```

Create, Drop, Alter Table Syntax

SQL CREATE TABLE STATEMENT

```
CREATE TABLE table_name( column1 datatype, column2 datatype, column3  
datatype, ..... , columnN datatype, PRIMARY KEY( one or more columns ) );
```

SQL DROP TABLE STATEMENT

```
DROP TABLE table_name;
```

SQL TRUNCATE TABLE STATEMENT

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE STATEMENT

```
ALTER TABLE table_name{ADD|DROP|MODIFY}column_name{data_type};
```

SQL ALTER TABLE STATEMENT (RENAME)

```
ALTER TABLE table_name RENAME TO new_table_name;
```

Insert, Update, Delete Syntax

SQL INSERT INTO STATEMENT

```
INSERT INTO table_name( column1, column2....columnN) VALUES ( value1,  
value2....valueN);
```

SQL UPDATE STATEMENT

```
UPDATE table_name SET column1 = value1, column2 =  
value2....columnN=valueN  
[ WHERE CONDITION ];
```

SQL DELETE STATEMENT

```
DELETE FROM table_name WHERE {CONDITION};
```

Select Statement Syntax

SQL SELECT STATEMENT

```
SELECT column1, column2....columnN FROM table_name;
```

SQL DISTINCT CLAUSE

```
SELECT DISTINCT column1, column2....columnN FROM  
table_name;
```

SQL WHERE CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
CONDITION;
```

SQL AND/OR CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
CONDITION-1 {AND|OR} CONDITION-2;
```

Select Statement Syntax

SQL IN CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
column_name BETWEEN val-1 AND val-2;
```

SQL LIKE CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
column_name LIKE { PATTERN };
```

SQL ORDER BY CLAUSE

```
SELECT column1, column2....columnN FROM table_name WHERE  
CONDITION ORDER BY column_name {ASC|DESC};
```

Select Statement Syntax

SQL GROUP BY CLAUSE

```
SELECT SUM(column_name) FROM table_name WHERE CONDITION  
GROUP BY column_name;
```

SQL COUNT CLAUSE

```
SELECT COUNT(column_name) FROM table_name WHERE CONDITION;
```

SQL HAVING CLAUSE

```
SELECT SUM(column_name) FROM table_name WHERE CONDITION  
GROUP BY column_name HAVING (arithmeticfunction condition);
```

Create and Drop Index Syntax

- **SQL CREATE INDEX Statement :**

```
CREATE UNIQUE INDEX index_name ON table_name( column1,  
column2,...columnN);
```

- **SQL DROP INDEX STATEMENT**

```
ALTER TABLE table_name DROP INDEX index_name;
```

- **SQL DESC Statement :**

```
DESC table_name;
```

Commit and Rollback Syntax

SQL COMMIT STATEMENT

COMMIT;

SQL ROLLBACK STATEMENT

ROLLBACK;

Inner Join Syntax

- The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an **EQUIJOIN**.
- The **INNER JOIN** creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

SYNTAX:

- The basic syntax of **INNER JOIN** is as follows:

```
SELECT table1.column1, table2.column2...FROM table1INNER JOIN table2ON  
table1.common_filed = table2.common_field;
```

Left Join Syntax

- The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.

- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

SYNTAX:

- The basic syntax of **LEFT JOIN** is as follows:

```
SELECT table1.column1, table2.column2...FROM table1LEFT JOIN table2ON  
table1.common_field = table2.common_field;
```

Right Join Syntax

- The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

SYNTAX:

- The basic syntax of **RIGHT JOIN** is as follows:

```
SELECT table1.column1, table2.column2...FROM table1RIGHT JOIN table2ON  
table1.common_field = table2.common_field;
```

Full Join Syntax

- The SQL **FULL JOIN** combines the results of both left and right outer joins.
- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

SYNTAX:

- The basic syntax of **FULL JOIN** is as follows:

```
SELECT table1.column1, table2.column2...FROM table1FULL JOIN table2ON  
table1.common_filed = table2.common_field;
```

Module - 2 [Manual Testing]

Agenda

- Fundamentals of Testing

- 7 Key Principles of Testing

- Fundamental Test Process(SDLC)

- Software Testing Levels

- Test Design Techniques

- Dynamic Testing Techniques

- Software Development Models

- Defect Management and Tracking

- Test Organization and Management

- Psychology of Testing

- Agile Testing

Fundamental al of Testing

What is Testing?(I)

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not.

This activity results in the actual, expected and difference between their results.

In simple words **testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements**

According to **ANSI/IEEE 1059** standard, Testing can be defined as A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Software testing is a process of executing a program or application with the intent of finding the software bugs.

What is Testing?(II)

Software Testing is a process used to identify the correctness, completeness, and quality of developed computer software.

- When asked, people often think that Testing only consists of running tests, i.e. executing the software
- Test execution is only a part of testing, but not all of the testing activities
- Test activities exist before and after test execution
- 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

What is Testing?(III)

- A Definition (and a Misconception)

- 'The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.'**

What is Testing?

Let's break the definition of Software testing into the following parts:

- **Process:** Testing is a process rather than a single activity.
- **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.
- **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.

What is Testing?(Cont...)

Let's break the definition of Software testing into the following parts:

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

Testing Activities

- Planning and control
- Choosing test conditions
- Designing test cases
- Checking results
- Evaluating completion criteria
- Reporting on the testing process and system under test
- Finalizing or closure (e.g. after a test phase has been completed)
- Testing also includes reviewing of documents (including source code) and static analysis

Test Objectives

- Finding defects

- Gaining confidence in and providing information about the level of quality.

- Preventing defects

- Both dynamic testing and static testing can be used as a means for achieving these objectives

- By designing tests early in the project life cycle it can help to prevent defects from being introduced into code

- Reviews of documents throughout the lifecycle (e.g. requirements and design) also help to prevent defects appearing in the code. More about this when we cover Static techniques

- They provide information in order to improve:

- The system to be tested

- The development and testing processes

- Live operations (e.g. how long it takes for a process to run)

Objectives and purpose

- Software testing makes sure that the testing is being done properly and hence the system is ready for use.
- Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues.
- It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

When to test ?

For the betterment, reliability and performance of an Information System, it is always better to involve the Testing team right from the beginning of the Requirement Analysis phase. The active involvement of the testing team will give the testers a clear vision of the functionality of the system by which we can expect a better quality and error-free product.

Once the Development Team-lead analyzes the requirements, he will prepare the System Requirement Specification, Requirement Traceability Matrix. After that he will schedule a meeting with the Testing Team (Test Lead and Tester chosen for that project). The Development Team-lead will explain regarding the Project, the total schedule of modules, Deliverable and Versions.

Why Testing is Necessary?

- Testing is necessary because we all make mistakes.
- Some of those mistakes are unimportant, but some of them are **expensive or dangerous**.
- We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.
- Since we assume that our work may have mistakes, hence we all need to check our own work.
- However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it.
- So we may not notice the flaws in what we have done.
- Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

Why Testing is Necessary?

Software Systems – Some context

Software Systems are now part of our everyday life

They are used almost everywhere, for example in:



- Banking and Financial institutions
 - Retail industry
 - Central and Local Government
 - Transport (e.g. Planes, Trains and Automobiles)
 - Medicine (Hospitals, research centres)
 - Home Entertainment
- We have all experienced
Software Systems failing!



Why Testing is Necessary?

Software Systems – When things go wrong

Software System Failures can lead to:

- Human Injury or Death
 - e.g. airplanes crashing
- Technological disasters
 - e.g. Missile Systems malfunctioning
- Legal action and associated costs
 - e.g. failure to meet contractual obligations
- Loss of face for suppliers and/or their customers;
 - e.g. mis-spelling company name on mail shots



When to start software testing?

- Testing is sometimes incorrectly thought as an after-the-fact activity; performed after programming is done for a product. Instead, testing should be performed at every development stage of the product .
- If we divide the lifecycle of software development into “Requirements Analysis”, “Design”, “Programming/Construction” and “Operation and Maintenance”, then testing should accompany each of the above phases. If testing is isolated as a single phase late in the cycle, errors in the problem statement or design may incur exorbitant costs.
- Not only must the original error be corrected, but the entire structure built upon it must also be changed. Therefore, testing should not be isolated as an inspection activity. Rather testing should be involved throughout the SDLC in order to bring out a quality product.

When to stop software testing ?

“When to stop testing” is one of the most difficult questions to a test engineer. The following are few of the common Test Stop criteria:

- All the high priority bugs are fixed.

- The rate at which bugs are found is too small.

- The testing budget is exhausted.

- The project duration is completed.

- The risk in the project is under acceptable limit.

Practically, we feel that the decision of stopping testing is based on the level of the risk acceptable to the management. The risk can be measured by Risk analysis but for small duration / low budget / low resources project, risk can be deduced by simply: –

- Measuring Test Coverage.

- Number of test cycles.

- Number of high priority bugs.

7 Key Principl e

General Testing Principles

1. Testing shows presence of Defects
2. Exhaustive Testing is Impossible!
3. Early Testing
4. Defect Clustering
5. The Pesticide Paradox
6. Testing is Context Dependent
7. Absence of Errors Fallacy

Testing shows presence of Defects

- Testing can show that defects are present, but cannot prove that there are no defects.
- Testing **reduces the probability of undiscovered defects** remaining in the software but, even if no defects are found, it is not a proof of correctness.
- We test to find Faults
- As we find more defects, the **probability of undiscovered defects** remaining in a system reduces.
- However Testing **cannot prove** that there are **no** defects present

Exhaustive Testing is Impossible!

- Testing everything including **all combinations of inputs and preconditions is not possible.**

- So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts.

- For example: In an application in **one screen there are 15 input fields**, each having 5 possible values, then to test all the valid combinations you would need **30 517 578 125 (5^{15}) tests.**

- This is very unlikely that the project timescales would allow for this number of tests.

- So, assessing and managing risk is one of the most important activities and reason for testing in any project.

- We have learned that we cannot test everything (i.e. all combinations of inputs and pre-conditions).

- That is we must **Prioritise** our testing effort using a **Risk Based Approach.**

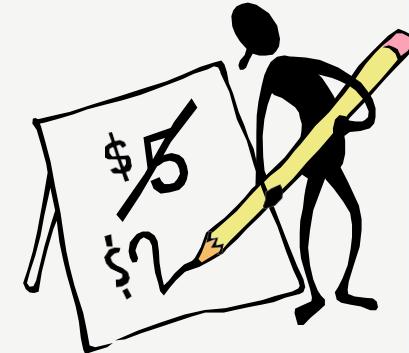
Why do not Testing Everything?

- Exhaustive testing of complex software applications:

- requires enormous resources
- is too expensive
- takes too long

- It is therefore impractical

- Need an alternative that is pragmatic, affordable, timely and provides results



Why do not Testing Everything?

Examples:

System has 20 screens
Average 4 menus / screen
Average 3 options / menu
Average of 10 fields / screen
2 types of input per field
Around 100 possible values

Approximate total for exhaustive testing
 $20 \times 4 \times 3 \times 10 \times 2 \times 100 = 480,000$ tests



Test length = 1 sec then test duration = 17.7 days
Test length = 10 sec then test duration = 34 weeks
Test length = 1 min then test duration = 4 years
Test length = 10 mins then test duration = 40 years!

Early Testing

- Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.
- Testing activities should start as **early** as possible in the development life cycle
- These activities should be focused on defined objectives – outlined in the Test Strategy
- Remember from our Definition of Testing, that Testing doesn't start once the code has been written!



Defect Clustering

- A small number of modules contain most of the defects discovered during pre-release testing, or are responsible for the most operational failures.
- Defects are not evenly spread in a system
- They are ‘clustered’
- In other words, most defects found during testing are usually confined to a small number of modules
- Similarly, most operational failures of a system are usually confined to a small number of modules
- An important consideration in test prioritisation!



Pesticide Paradox

If the same tests are repeated over and over again, eventually the same set of **test cases will no longer find any new defects**.

To overcome this “pesticide paradox”, the test cases need to be **regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects**.

Testing identifies bugs, and programmers respond to fix them

As bugs are eliminated by the programmers, the software improves

As software improves the effectiveness of previous tests erodes



Pesticide Paradox

Therefore we must learn, create and use new tests based on new techniques to catch new bugs

N.B It's called the "pesticide paradox" after the agricultural phenomenon, where bugs such as the boll weevil build up tolerance to pesticides, leaving you with the choice of ever-more powerful pesticides followed by ever-more powerful bugs or an altogether different approach.' – Beizer 1995



Testing is Context Dependent

- Testing is basically context dependent.
- Testing is done differently in different contexts
- **Different kinds of sites are tested differently.**
- For example
 - **Safety - critical software is tested differently from an e-commerce site.**
- Whilst, Testing can be 50% of development costs, in NASA's Apollo program it was 80% testing
- 3 to 10 failures per thousand lines of code (KLOC) typical for commercial software
- 1 to 3 failures per KLOC typical for industrial software
- 0.01 failures per KLOC for NASA Shuttle code!
- Also different industries impose different testing standards

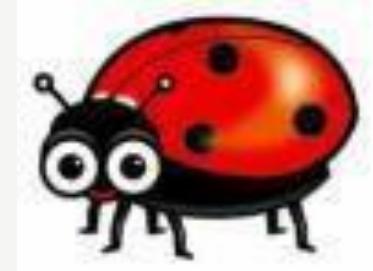
Absence of Errors Fallacy

- If the system built is unusable and does not fulfill the user's needs and expectations then finding and fixing defects does not help.
- If we build a system and, in doing so, find and fix defects
 - It doesn't make it a **good** system
- Even after defects have been resolved it may still be **unusable** and/or does not fulfil the users' **needs and expectations**

Error , Defect , Bug and Failure

Causes of Software Failure

- A Human can make an Error
- An Error is '**A Human Action that produces an Incorrect Result'**
- The Error can cause a Defect
- A Defect is '**A flaw in a component or system that can cause the component or system to fail to perform its required function'**
- A Defect can be in the Software, System or in a Document



Errors, Defects and Failures

Defects occur because human beings are fallible

Also because of:

- time pressure
- complex code
- complex infrastructure
- changed technologies
- and/or many system interactions

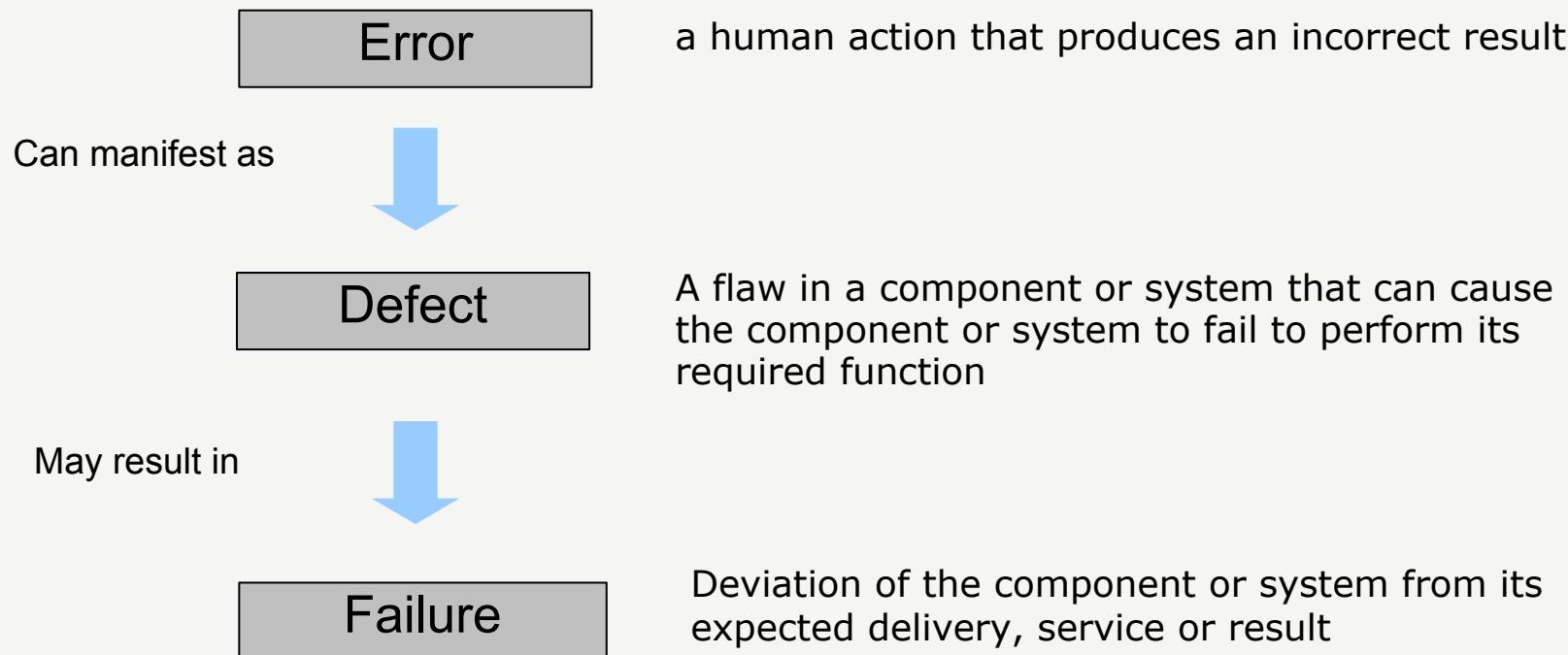
A **Defect** may result in a **Failure**

A **Failure** is a '**Deviation of the component or system from its expected delivery, service or result**'

Failures can be caused by environmental conditions as well

- E.g. radiation, magnetism, electronic fields
- Pollution can cause faults in firmware or influence the execution of software by changing hardware conditions.

Errors, Defects and Failures



Errors, Defects and Failures

“A mistake in coding is called error, error found by tester is called defect, defect accepted by development team then it is called bug, build does not meet the requirements then it is failure”

Error: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. This can be a misunderstanding of the internal state of the software, an oversight in terms of memory management, confusion about the proper way to calculate a value, etc.

Errors, Defects and Failures

Failure: The inability of a system or component to perform its required functions within specified performance requirements. See: bug, crash, exception, and fault.

Bug: A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is terminology of Tester.

Fault: An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. See: bug, defect, error, exception.

Defect: Commonly refers to several troubles with the software products, with its external behavior or with its internal features.

Types of Errors

- User Interface Errors
- Error Handling
- Boundary related errors
- Calculation errors
- Initial and Later states
- Control flow errors
- Errors in Handling or Interpreting Data
- Race Conditions
- Load Conditions
- Hardware
- Source, Version and ID Control
- Testing Errors

The Role of Testing

Rigorous testing of systems and documentation can:

- reduce the risk of problems occurring in an operational environment
- contribute to the quality of the software system

How?

- By finding and correcting defects before the system is released for operational use
- Software testing may also be required to meet contractual or legal requirements, or industry-specific standards

**Qu
alit
y**

Quality

- Quality - '**The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations'**'

- Defects covering:

- functional software requirements and characteristics
- and non-functional software requirements and characteristics (e.g. reliability, usability, efficiency, portability and maintainability)

- Testing can give confidence in the Quality of the software if it finds few or no defects

- Quality software is reasonably **bug or defect free**, delivered on time and within budget, meets requirements and/or expectations, and is maintainable.

- ISO 8402-1986 standard defines quality as "**the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs.**"

Quality(Cont...)

- Key aspects of quality for the customer include:

- Good design – looks and style
- Good functionality – it does the job well
- Reliable – acceptable level of breakdowns or failure
- Consistency
- Durable – lasts as long as it should
- Good after sales service
- Value for money

Following are two cases that demonstrate the importance of software quality

- Failure due to error in a transfer of information between a team in Colorado and a team in California
- One team used English units (e.g., inches, feet and pounds) while the other used metric units for a key spacecraft operation.

**Ri
sk**

Risk

- A properly designed test that passes, reduces the overall level of Risk in a system
- Risk – **'A factor that could result in future negative consequences; usually expressed as impact and likelihood'**
- When testing does find defects, the Quality of the software system increases when those defects are fixed
- The Quality of systems can be improved through Lessons learned from previous projects
- Analysis of root causes of defects found in other projects can lead to Process Improvement
- Process Improvement can prevent those defects reoccurring
- Which in turn, can improve the Quality of future systems
- Testing should be integrated as one of the Quality assurance activities

Types of Risk

A Risk could be any future event with a negative consequence

You

need to identify the risks associated with your project

Risks are of two types

- Project Risks
- Product Risk

Types of Risk Examples

Example of **Project risk** is Senior Team Member leaving the project abruptly.

- Every risk is assigned a likelihood i.e. chance of it occurring, typically on a scale of 1 to 10. Also the impact of that risk is identified on a scale of 1-10 .
- But just identifying the risk is not enough. You need to identify mitigation. In this case mitigation could be Knowledge Transfer to other team members & having a buffer tester in place

Example of **product risks** would be Flight Reservation system not installing in test environment

- Mitigation in this case would be conducting a smoke or sanity testing. Accordingly you will make changes in your scope items to include sanity testing

Test Organiza tion

Who does Testing?

- It depends on the process and the associated stakeholders of the project(s).
- In the IT industry, large companies have a team with responsibilities to **evaluate the developed software in the context of the given requirements.**
- Moreover, **developers also conduct testing which is called Unit Testing.**
- In most cases, following professionals are involved in testing of a system within their respective capacities:

- **Software Tester**
- **Software Developer**
- **Project Lead/Manager**
- **End User**

Different companies have different designations for people who test the software on the **basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, and QA Analyst** etc.

- It is not possible to test the software at any time during its cycle.
- The next two sections state when testing should be started and when to end it during the SDLC.

What do Tester?

- Make up the majority of the resources in the testing group.
- May be specialists in a particular area
 - Automation
 - Performance
 - Usability
 - Security
- Or alternatively may work more generally doing:
 - Test Analysis
 - Test Design
 - Test Execution
 - Test Environment management
 - Test Data management
- Typically testers at the component level are developers
- At the Acceptance level testers are typically business experts or users or operators (for operational acceptance testing)

Role of Software Tester

Apart from exposing faults (“bugs”) in a software product confirming that the program meets the program specification, as a test engineer you need to create test cases, procedures, scripts and generate data.

You execute test procedures and scripts, analyze standards and evaluate results of system/integration/regression testing. You also...

- Speed up development process by identifying bugs at an early stage (e.g. specifications stage)
- Reduce the organization's risk of legal liability
- Maximize the value of the software
- Assure successful launch of the product, save money, time and reputation of the company by discovering bugs and design flaws at an early stage before failures occur in production, or in the field
- Promote continual improvement



Test Planning Template Or Test

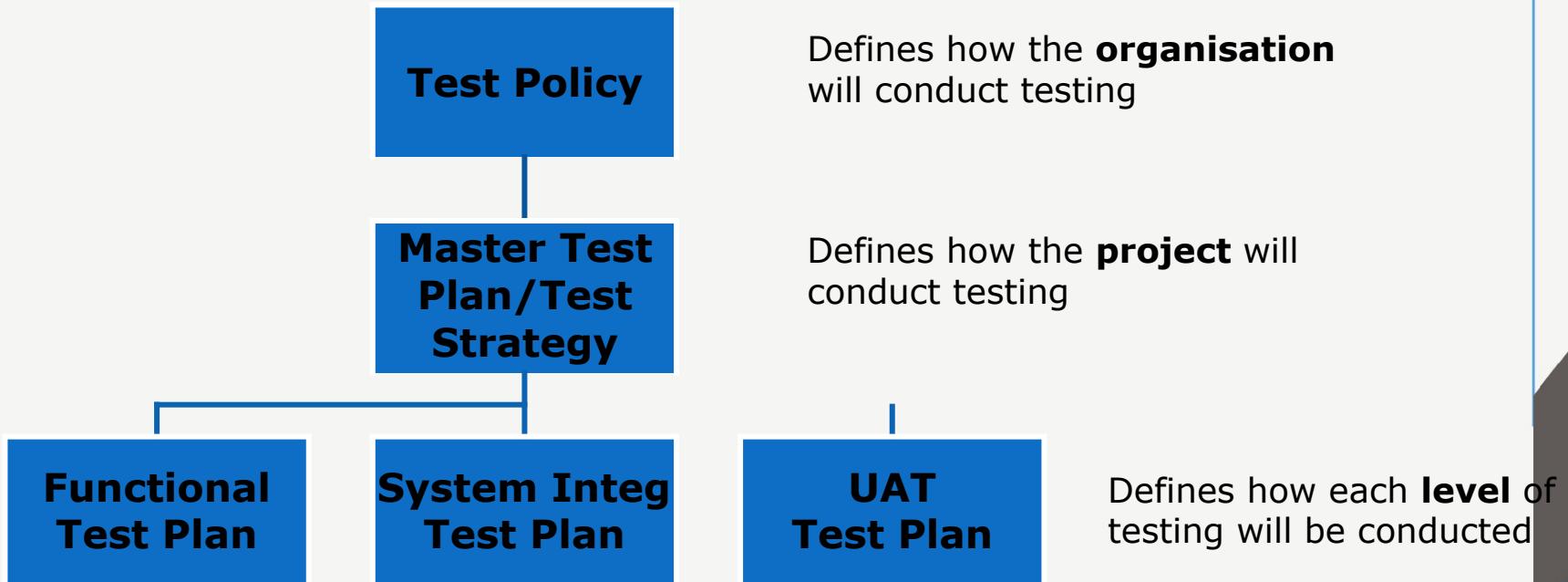
Test Planning

A document describing the scope, approach, resources and schedule of intended test activities

- Determining the **scope and risks, and identifying** the objectives of testing.
- Defining the overall approach of testing (the test strategy), including the definition of the test levels and entry and exit criteria.
- Integrating and coordinating the testing activities into the software life cycle activities:
 - **acquisition, supply, development, operation and maintenance.**
- Making decisions about what to test, what roles will perform the test activities, how the test activities should be done, and how the test results will be evaluated?
- Scheduling test analysis and design activities.
- Scheduling test implementation, execution and evaluation.
- Assigning resources for the different activities defined
 - **Defining the amount, level of detail, structure and templates for the test documentation.**

Test Plan & Strategy

- All projects require a set of plans and strategies which define how the testing will be conducted.
- There are number of levels at which these are defined:



Test Planning Factors

- Factors which affect test planning

- The organisation's test policy
- Scope of the testing being performed
- Testing objectives
- Project Risks – e.g. business, technical, people
- Constraints – e.g. business imposed, financial, contractual etc
- Criticality (e.g. system/component level)
- Testability
- Availability of resources

- Test plans are continuously refined

- As more information becomes available
- As new risks arise or others are mitigated
- Not set in concrete, but changes must be carefully managed

Test Planning Activities

- **Approach:** Defining the overall approach of testing (the test strategy), including the definition of the test levels and entry and exit criteria.
- **Integrating and coordinating the testing activities into the software life cycle activities:** acquisition, supply, development, operation and maintenance.
- Making decisions about:
 - **what** to test
 - **who** do testing? i.e. what roles will perform the test activities
 - **when** and how the test activities should be done and when they should be stopped (exit criteria – see next slides)
 - **how** the test results will be evaluated
- Assigning resources for the different tasks defined.
- **Test ware:** Defining the amount, level of detail, structure and templates for the test documentation.
- Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues.
- **Process:** Setting the level of detail for test procedures in order to provide enough information to support reproducible test preparation and execution.

Exit Criteria

How do we know when to stop testing?

- Run out of time?
- Run out of budget?
- The business tells you it went live last night!
- Boss says stop?
- All defects have been fixed?
- When our exit criteria have been met?

Purpose of exit criteria is to define when we STOP testing either at the:

- End of all testing – i.e. product Go Live
- End of phase of testing (e.g. hand over from System Test to UAT)

Exit Criteria typically measures:

- Thoroughness measures, such as coverage of requirements or of code or risk coverage
- Estimates of defect density or reliability measures. (e.g. how many defects open by category)
- Cost.
- Residual Risks, such as defects not fixed or lack of test coverage in certain areas.
- Schedules - such as those based on time to market.

QA vs QC vs Testing

S.N.	Quality Assurance	Quality Control	Testing
1	Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements.	Activities which ensure the identification of bugs/error/defects in the Software.
2	Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
3	Process oriented activities.	Product oriented activities.	Product oriented activities.
4	Preventive activities.	It is a corrective process.	It is a preventive process.
5	It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

How much Testing is Enough?

Deciding how much testing is enough should take account of:

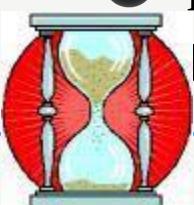
- the level of Risk
- project constraints such as time and budget

Risks should be evaluated at the Business Level, Technological Level, Project Level and Testing Level

Risks are also used to decide where to start testing and where more testing is needed

Risk considerations can include:

- financial implication of software being released that is unreliable
(support costs / possible legal action)
- software being delivered late to market
- potential loss of Life (safety critical systems)
- potential loss of face (may have implications as well)



How much Testing is Enough?

- Risk analysis should be used to determine what to test in each component and just as importantly what not to test

- For example, an unacceptable risk would say we must test, an acceptable one perhaps not to test

- Testing is a risk-control activity that provides feedback to the stakeholders

- With this feedback the stakeholders can make informed decisions about the release of the software (or system) being tested

- More about Risks later in the Course

- **Exit criteria** is used to determine when testing at any stage is complete

The set of generic and specific conditions, agreed upon with the stakeholders, for permitting a process to be officially completed

- Exit criteria may be defined in terms of :

- Thoroughness – i.e. coverage or requirements
- cost or time constraints
- percentage of tests run without incident
- number of faults remaining

Testing v/s Debugging

The responsibility for each activity is very different, i.e.

- Testers test
- Developers debug

Testing

- It involves the identification of bug/error/defect in the software without correcting it.
- Normally professionals with a Quality Assurance background are involved in the identification of bugs. Testing is performed in the testing phase.
- Testing can show failures that are caused by defects

Debugging

- It involves identifying, isolating and fixing the problems/bug. Developers who code the software conduct debugging upon encountering an error in the code.
- Debugging is the part of White box or Unit Testing.
- Debugging can be performed in the development phase while conducting Unit Testing or in phases while fixing the reported bugs
- Debugging identifies the cause of a defect, repairs the code and checks that the defect has been fixed correctly

Test Development Process

Introduction(Cont...)

● **Test Analysis**

● **Test Plan/Strategy**

● **Test Script/Test Step/Test Procedure**

● **Test Scenario**

● **Test Case**

● **Test Condition**

● **Test Procedure Specification**

● **Traceability**

● **Tractability Matrix**

Test Analysis

- Test analysis is the process of looking at something that can be used to derive test information. This basis for the tests is called the **test basis**.
- The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces.
- We can use the test basis documents to understand what the system should do once built. The test basis includes whatever the tests are based on. Sometimes tests can be based on experienced user's knowledge of the system which may not be documented.

Test Analysis(Cont...)

From testing perspective we look at the test basis in order to see what could be tested. These are the test conditions. A **test condition** is simply something that we could test.

While identifying the test conditions we want to identify as many conditions as we can and then we select about which one to take forward and combine into test cases. We could call them **test possibilities**.

The test conditions that are chosen will depend on the test strategy or detailed test approach. For example, they might be based on risk, models of the system, etc.

Once we have identified a list of test conditions, it is important to prioritize them, so that the most important test conditions are identified.

Test conditions can be identified for **test data** as well as for test inputs and test outcomes, for example, different types of record, different sizes of records or fields in a record.

High Level Requirement

#	Name	Description
	Search	
100	DVD name	The system shall provide the ability for the user to search by a DVD Name
101	Actor name	The system shall provide the ability for the user to search by a Actor name
102	Categories within search	The system shall provide the ability for the user to search by categories within search
	User account	
200	Login	
201	Logout	
202	Forgot user name and password	
203	Create user account	

Test Script

- **A set of sequential instruction that detail how to execute a core business function**

- One script is written to explain how to simulate each business scenario
- Written to a level of detail for which someone else (other than the script writer) would be able to easily execute
- Identifies the test condition that is being satisfied for each step, if applicable
- Identified the input/test data that should be entered for each transaction
- Identifies the expected results for each step, if applicable
- Should demonstrate how the system can support the HCA warehouse business processes

Test Script

- A test script in software testing is **a set of instructions that will be performed on the system under test to test that the system functions as expected.**

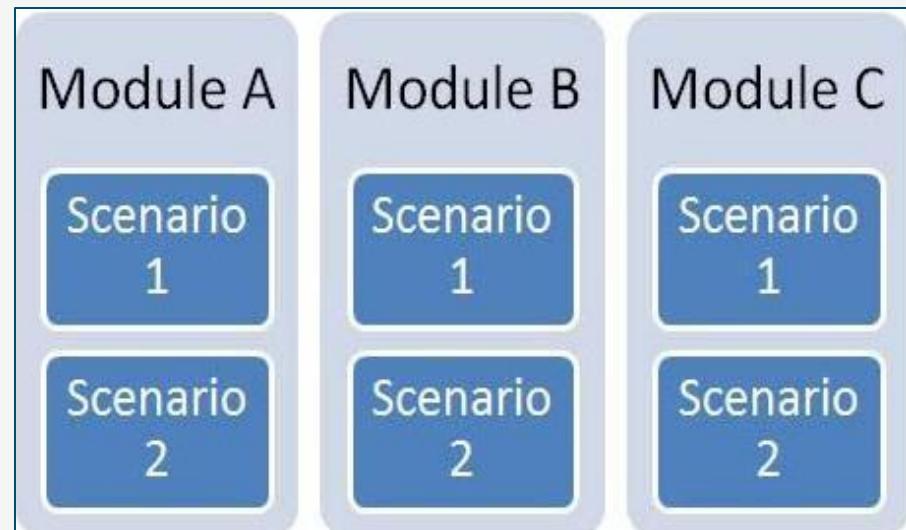
- There are various means for executing test scripts.

- **Manual Testing**
- **Automation Testing**

Test Scenario

A Scenario is any functionality that can be tested. It is also called Test Condition, or Test Possibility.

- Test Scenario is ‘What to be tested’
- Test scenario is nothing but test procedure.
- The scenarios are derived from use cases.
- Test Scenario represents a series of actions that are associated together.
- Scenario is thread of operations



Test Case

Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks.

- Test Case is ‘How to be tested’
- Test case consist of set of input values, execution precondition, expected Results and executed post-condition developed to cover certain test Condition.
- Test cases are derived (or written) from test scenario.
- Test Case represents a single (low level) action by the user.
- Test cases are set of input and output given to the System.

Test Case

Furthermore test cases are written to keep track of testing coverage of Software. Generally, there is no formal template which is used during the test case writing. However, following are the main components which are always available and included in every test case:

- Test case ID
- Product Module ID
- Product version (Optional)
- Revision history (Optional)
- Purpose/ Test Case Description
- Assumptions (Optional)
- Pre-Conditions(Optional)
- Test Steps
- Expected Outcome/Result
- Actual Outcome/Result
- Post Conditions(Pass/Fail)

Test Case(Cont...)

- The Step **# Identifies** the task sequence in the script

Action/Input Data

- The **Action Steps** details the task to be performed

- Write action steps in terms that support execution (action oriented)

- Level of detail should reflect core business function, not system navigation

- The **Input Data** describes what needs to be entered for a step (if applicable),
It's called **Test Data**.

- Include all search criteria (Ex. First name, last name)

Expected Results

- The Expected Results documents what results are anticipated for this step
- Include detail needed for business process (Ex. Required fields, external system interfaces)

Actual Results

- Where the “script executor” enters the result that occurred from this step
- Be specific

Test Case

- Developing test material can be split into two distinct stages:
 - Defining “what” needs to be tested
 - Defining “how” the system should be tested
- This process can vary from organisation to organisation, can be very formal or very informal with little documentation
- The more formal, the more repeatable the tests, but it does depend on the context of the testing being carried out
- The process of identifying test conditions and designing tests consists of the following steps:
 - **Identify and Defining Test Conditions**
 - **Specifying Test Cases**
 - **Specifying Test Procedures**
 - **Developing a Test Execution Schedule**

Test Conditions

- Test Conditions are binary statements which determine a system's fitness for purpose
- Defines **what** must be tested
- Sometimes referred to as a Test Item
- Grouped by Test Object or system function/process
- Test Requirement (Test basis) documentation is analysed to determine the Test Conditions
- Test Conditions are then cross referenced to one or more test cases for execution
- Not all Test Conditions are as important as others so each Test Condition is assigned a risk

Test Conditions(Cont...)

Test Conditions should be linked back to their source documents from which they are derived. This helps for two reasons:

- Impact Analysis
- Traceability

A Test Case defines **how** the system should be tested

They typically contain

- Input values
- Execution pre conditions
- Expected results (output, changes in state etc.)
- Post conditions
- Cross referenced test conditions

Remember expected results must be defined before execution

There can be many Test Cases developed to test a single Test Condition

Test Procedures Specification (Test Script)

- The Test Procedures Specification **specifies the sequence of actions for a test**, i.e. one or more Test Cases

- It is also known as a **Test Script**

- The Test Script can be manual or automated

- Contents of a Test Procedure are:

- Test procedure specification identifier
- Purpose
- Special requirements
- Procedure steps

Test Execution Schedule

- The Test Procedure Specifications (i.e. Test Scripts) are subsequently included in a Test Execution Schedule

- This schedule defines the order in which the test scripts are executed, when they are to be carried out and by whom

- The Execution schedule will also need to take account of:

- Regression Tests
- Prioritisation
- And technical and logical dependencies

Tractability

Test conditions should be able to be linked back to their sources in the test basis, this is known as traceability.

Traceability can be horizontal through all the test documentation for a given test level (e.g. system testing, from test conditions through test cases to test scripts) or it can be vertical through the layers of development documentation (e.g. from requirements to components).

Tractability Matrix

- To protect against changes you should be able to **trace back from every system component** to the original requirement that caused its presence.

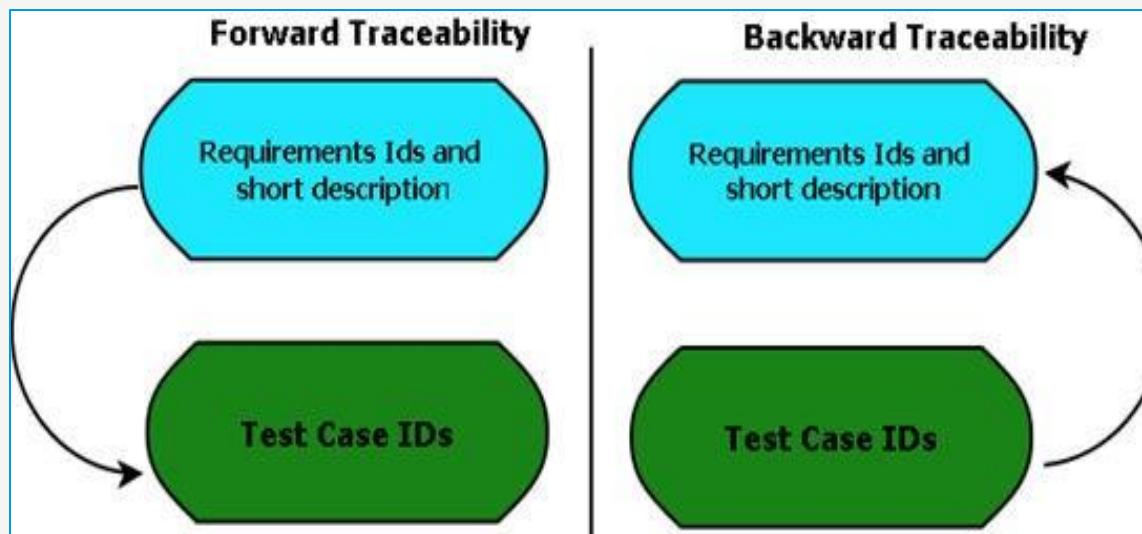
- A **software process** should help you keeping the virtual table up-to-date.

- Simple technique may be quite valuable (naming convention)

	Comp 1	Comp 2	:	:	:	:	:	:	Comp m
Req 1			x		x				
Req 2	x								x
...									
...		x			x		x		
...									
...		x							
...				x					
...								x	
Req n									

Types of Traceability Matrix

- Forward Traceability – Mapping of Requirements to Test cases
- Backward Traceability – Mapping of Test Cases to Requirements
- Bi-Directional Traceability - A Good Traceability matrix is the References from test cases to basis documentation and vice versa.



Pros of Traceability Matrix

- Make obvious to the client that the software is being developed as per the requirements.
- To make sure that all requirements included in the test cases
- To make sure that developers are not creating features that no one has requested
- Easy to identify the missing functionalities.
- If there is a change request for a requirement, then we can easily find out which test cases need to update.
- The completed system may have “Extra” functionality that may have not been specified in the design specification, resulting in wastage of manpower, time and effort.

Cons of Traceability Matrix

- No traceability or Incomplete Traceability Results into:
- Poor or unknown test coverage, more defects found in production
- It will lead to miss some bugs in earlier test cycles which may arise in later test cycles. Then a lot of discussions arguments with other teams and managers before release.
- Difficult project planning and tracking, misunderstandings between different teams over project dependencies, delays, etc

STLC – Software Testing Life Cycle

Stages of STLC

1. Test Planning and Controlling
2. Test Analysis and Design
3. Test Implementation and Execution
4. Evaluating Exit Criteria and Reporting
5. Test Closure Activities



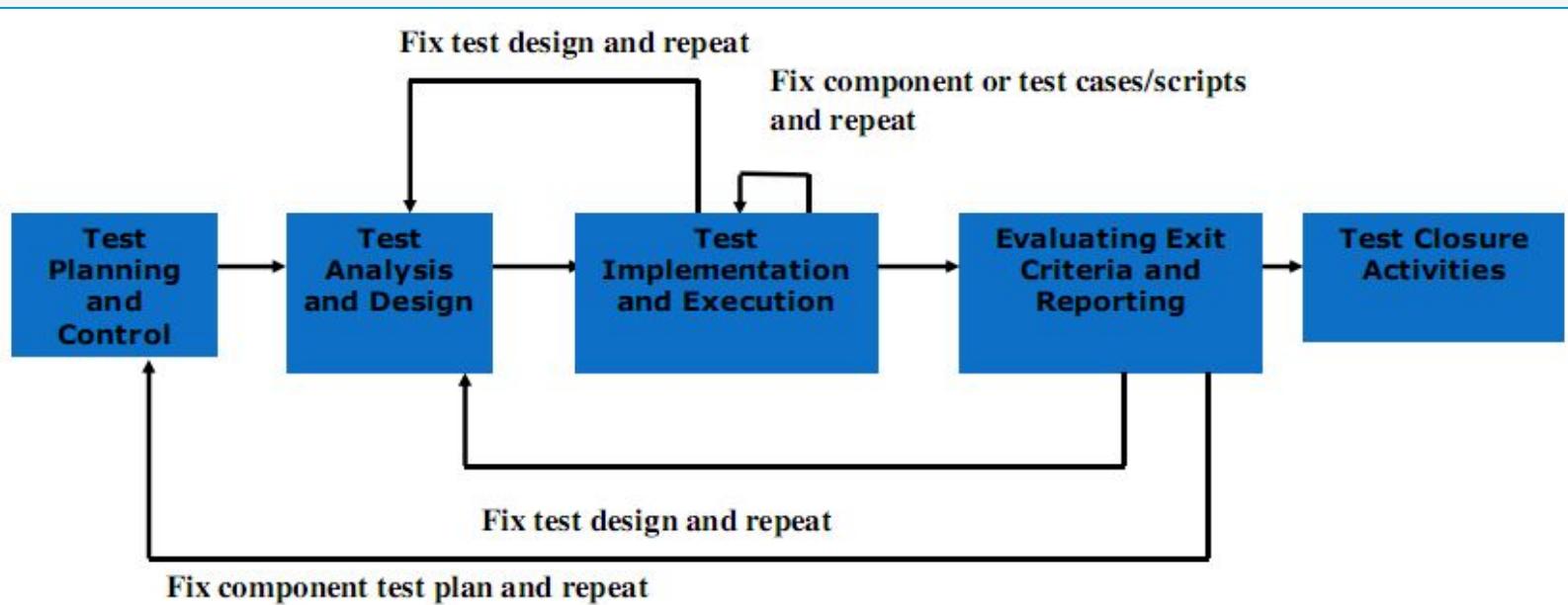
Fundamental Test

Process(STLC)

- The process always starts with planning and ends with test closure activities

- Each phase may have to be executed a number of times in order to fulfil exit or completion criteria

- Although logically sequential, the activities in the process may overlap or take place concurrently



Test Planning & Controlling

- Specifies how the test strategy and project test plan

A document describing the scope, approach, resources and schedule of intended test activities apply to the software under test.

- Principally:

- verify the mission
- define the Test objectives
- Specify the Test Activities required to meet the mission and objectives

- Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues.

- Setting the level of detail for test procedures in order to provide enough information to support reproducible test preparation and execution.

Test Planning Major Task

- To determine the scope and risks and identify the objectives of testing.
- To determine the test approach.
- To implement the test policy and/or the **test strategy**.
- To determine the required test resources like people, test environments, PCs, etc.
- To schedule test analysis and design tasks, test implementation, execution and evaluation.
- To determine the **Exit criteria** we need to set criteria such as **Coverage criteria**. (**Coverage criteria** are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

Test Controlling Major Task

- To measure and analyze the results of reviews and testing.
- To monitor and document progress, test coverage and exit criteria.
- To provide information on testing.
- To initiate corrective actions.
- To make decisions.

Test Analysis and Design

General testing objectives are transformed into tangible Test Conditions (An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element) and Test Designs (A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high level test cases).

Tests should be designed using the test design techniques selected in the test planning activity.

Test Analysis & Design Major

Task

- To review the **test basis**. (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
- To identify **Test Conditions/Requirements and required test data** from analysis of test items
- To design the tests (note – the detail, in the form of a Test Case, is developed in the next stage)
- To **evaluate testability of the requirements and system**
- To **design the test environment set-up**
- To **Identify any required infrastructure and tools**

Test Implementation & Execution

- During test implementation and execution, we take the test conditions into **test cases** and **procedures** and other **test ware** such as scripts for automation, the test environment and any other test infrastructure.
(**Test cases** are a set of conditions under which a tester will determine whether an application is working correctly or not.)
- (**Test ware** is a term for all utilities that serve in combination for testing software like scripts, the test environment and any other test infrastructure for later reuse.)
- Test Conditions are transformed into Test Cases and Test ware
- The test environment is created



Test Implementation Major

Task

To **develop and prioritize our test cases by using techniques and create test data** for those tests. We also write some instructions for carrying out the tests which is known as **test procedures**.

We may also need to automate some tests using **test harness** and automated tests scripts. (A **test harness** is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)

To **create test suites from the test cases for efficient test execution**. (**Test suite** is a collection of test cases that are used to test a software program to show that it has some specified set of behaviors. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)

To **implement and verify the environment**.



Test Execution Major



Task

- To **execute test suites and individual test cases following the test procedures.**
- To **re-execute the tests that previously failed in order to confirm a fix.** This is known as **confirmation testing or re-testing.**
- To log the outcome of the **test execution and record the identities and versions of the software under tests.**
- The **test log** is used for the audit trial. (A **test log** is nothing but, what are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
- To **compare actual results with expected results.**



Test Execution Major



Task

- Where there are **differences between actual and expected results, it report discrepancies as Incidents.**
- Analyse incidents to **establish root cause**
- The test coverage levels achieved for those measures specified as **test completion criteria should be recorded.**

Evaluating Exit Criteria & Reporting

- Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the **“enough testing”**. These criteria vary from project to project and are known as **exit criteria**.
- Test execution is assessed against the objectives defined in Test Planning
- This should be done for each **Test Level** (i.e. test stage)
 - **A group of test activities that are organized and managed together.**
- Exit criteria come into picture, when:
 - Maximum test cases are executed with certain pass percentage.
 - Bug rate falls below certain level.
 - When achieved the deadlines.



Evaluating Exit Criteria Major

Task

- To check the test logs against the exit criteria specified in test planning.
- To assess if more test are needed or if the exit criteria specified should be changed.
- To write a test summary report for stakeholders.
- If the exit criteria has not been met
 - Assess if more tests are needed
 - Assess which test activities may need to be repeated

Test Closure Activities

- Collect data from completed test activities to consolidate experience, Testware, facts and numbers
- Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:
 - When all the information has been gathered which are needed for the testing.
 - When a project is cancelled.
 - When some target is achieved.
 - When a maintenance release or update is done.



Test Closure Activities Major

Task

- To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.
- To check that incident reports status are up-to-date (e.g. Closed)
- To ensure all Incident reports have associated change records
- To record acceptance of the system
- To finalize and archive test ware such as scripts, test environments, etc. for later reuse.
- To handover the test ware to the maintenance organization. They will give support to the software.
- To evaluate how the testing went and learn lessons for future releases and projects.

Psychology of Testing

Relation Between Tester & Developer

- The testing and reviewing of the applications are different from the analyzing and developing of it.
- By this we mean to say that if we are building or developing applications we are working positively to solve the problems during the development process and to make the product according to the user specification.
- However while testing or reviewing a product we are looking for the defects or failures in the product.
- Thus building the software requires a different mindset from testing the software.

Relation Between Tester & Developer

It does not mean that the tester cannot be the programmer, or that the programmer cannot be the tester, although they often are separate roles. In fact programmers are the testers.

They always test their component which they built. While testing their own code they find many problems so the programmers, architect and the developers always test their own code before giving it to anyone.

However we all know that it is difficult to find our own mistakes. So, programmers, architect, business analyst depend on others to help test their work.

This other person might be some other developer from the same team or the Testing specialists or professional testers. Giving applications to the testing specialists or professional testers allows an independent test of the system.

Self Testing & Independent Testing

This degree of independence avoids author bias and is often more effective at finding defects and failures

There are several levels of independence in software testing which are listed here from the lowest level of independence to the highest:

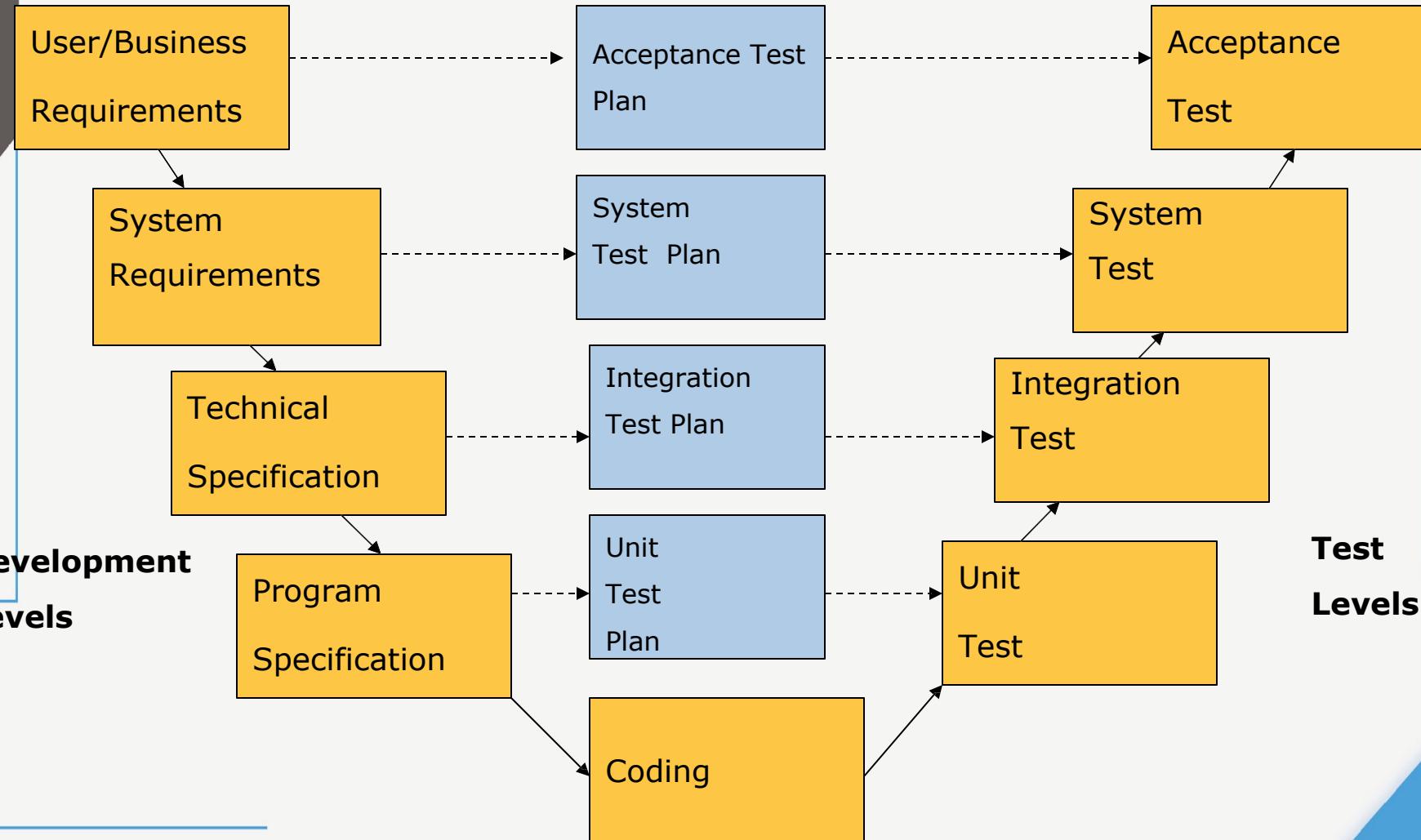
- Tests by the person who wrote the item.
- Tests by another person within the same team, like another programmer.
- Tests by the person from some different group such as an independent test team.
- Tests by a person from a different organization or company, such as outsourced testing or certification by an external body.

Software Development Model

Agenda

- V – Model (Verification and Validation Model)
- RAD Model (Rapid Access Development Model)

V-Model Design



V-Model Description

The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. **It is also known as Verification and Validation model.**

Under V-Model, the corresponding testing phase of the development phase is planned in parallel.

So there are Verification phases on one side of the .V. and Validation phases on the other side. Coding phase joins the two sides of the V- Model.

Although variants of the V-model exist, a common type of V-model uses four test levels, corresponding to the four development levels.

The four levels used in this syllabus are:

- **Component (unit) testing**
- **Integration testing**
- **System testing**
- **Acceptance testing**

Verification Phase

Business Requirement Analysis: This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

System Design (System Requirement): Once you have the clear and detailed product requirements, it's time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.

Verification Phase(Cont...)

Architectural Design (Technical Specification): Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design (Program Specification): In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Code Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phase

Unit Testing: Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing: Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing: System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.

Acceptance Testing: Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

V-Model Application

- V- Model application is almost same as waterfall model, as both the models are of sequential type.
- Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes.
- This model is used in the medical development field, as it is strictly disciplined domain.
- Following are the suitable scenarios to use V-Model:
 - Requirements are well defined, clearly documented and fixed.
 - Product definition is stable.
 - Technology is not dynamic and is well understood by the project team.
 - There are no ambiguous or undefined requirements.
 - The project is short.

V-Model Pros & Cons

Pros of V-Model

- This is a highly disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

Cons of V-Model

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality
- No working software is produced until late during the life cycle.

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 requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business 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 fulfills 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	<ul style="list-style-type: none"> • Reviews • Walkthroughs • Inspections 	<ul style="list-style-type: none"> • Testing

RAD Model

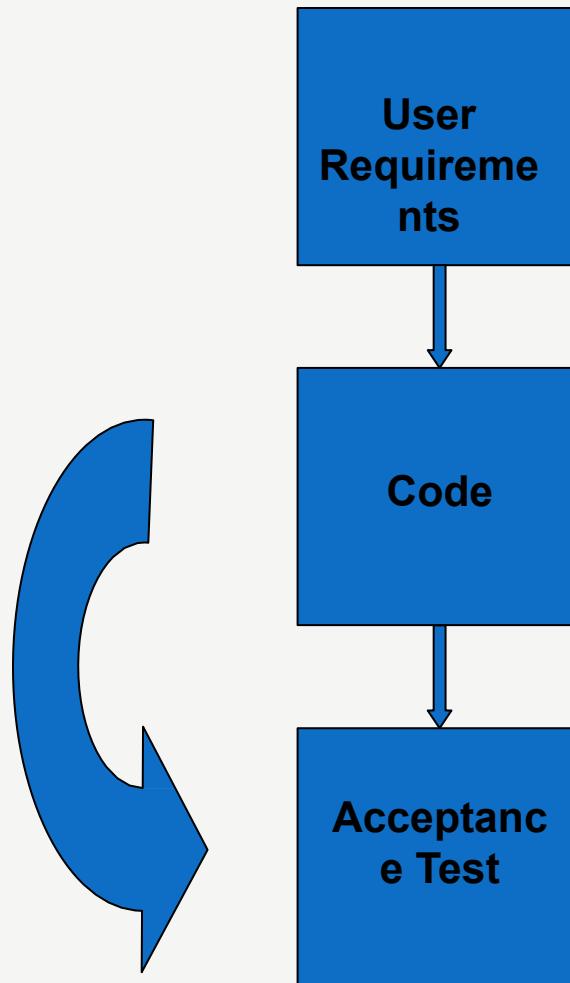
- The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.
- Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.
- Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.
- In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

RAD Model(Cont...)

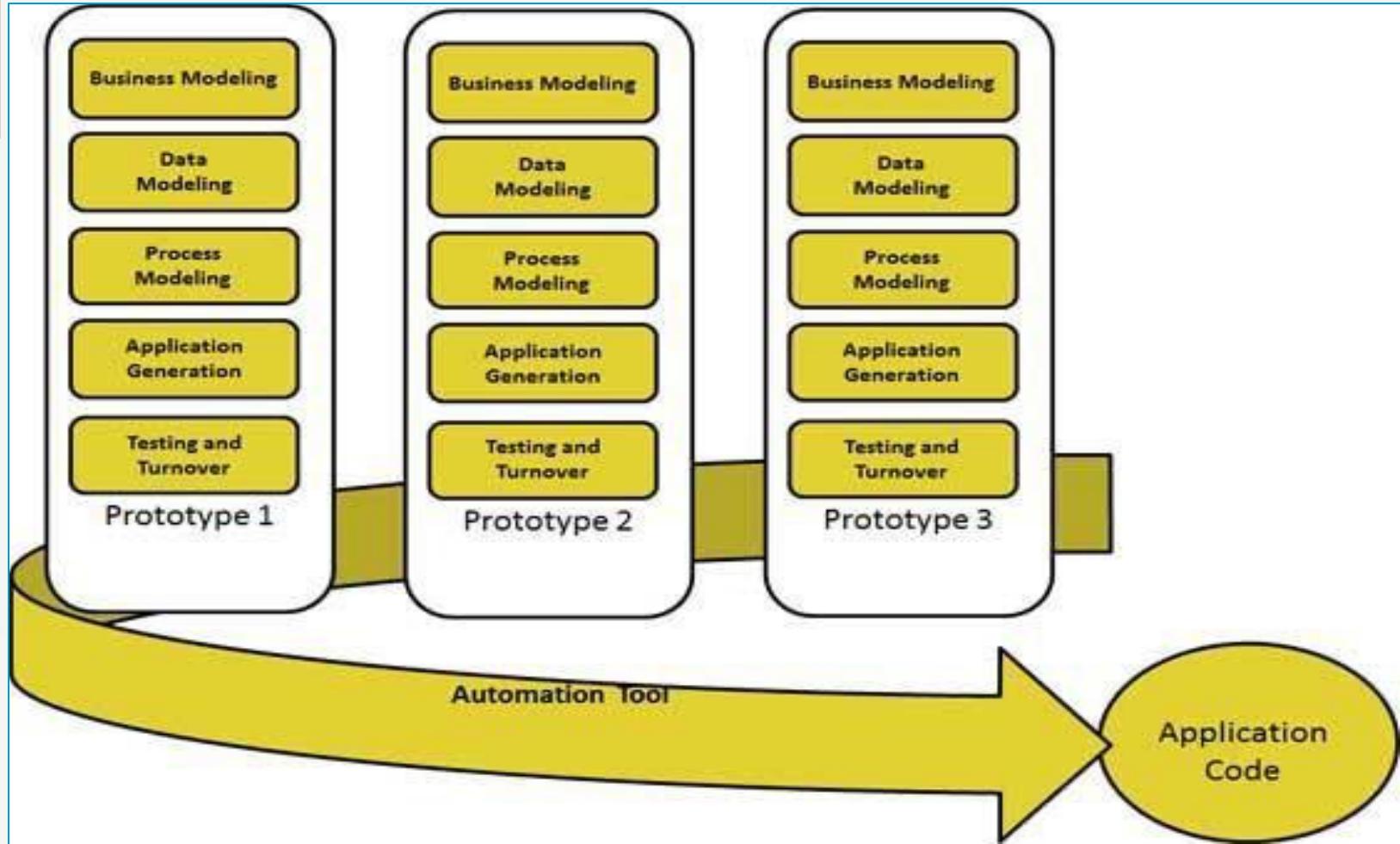
- Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

- RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

- The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.



RAD Model Design



RAD Model Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a system can be modularized to be delivered in incremental manner.
- It should be used if there's high availability of designers for modeling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the course of the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

Pros of RAD Model

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in short time.
- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Cons of RAD Model

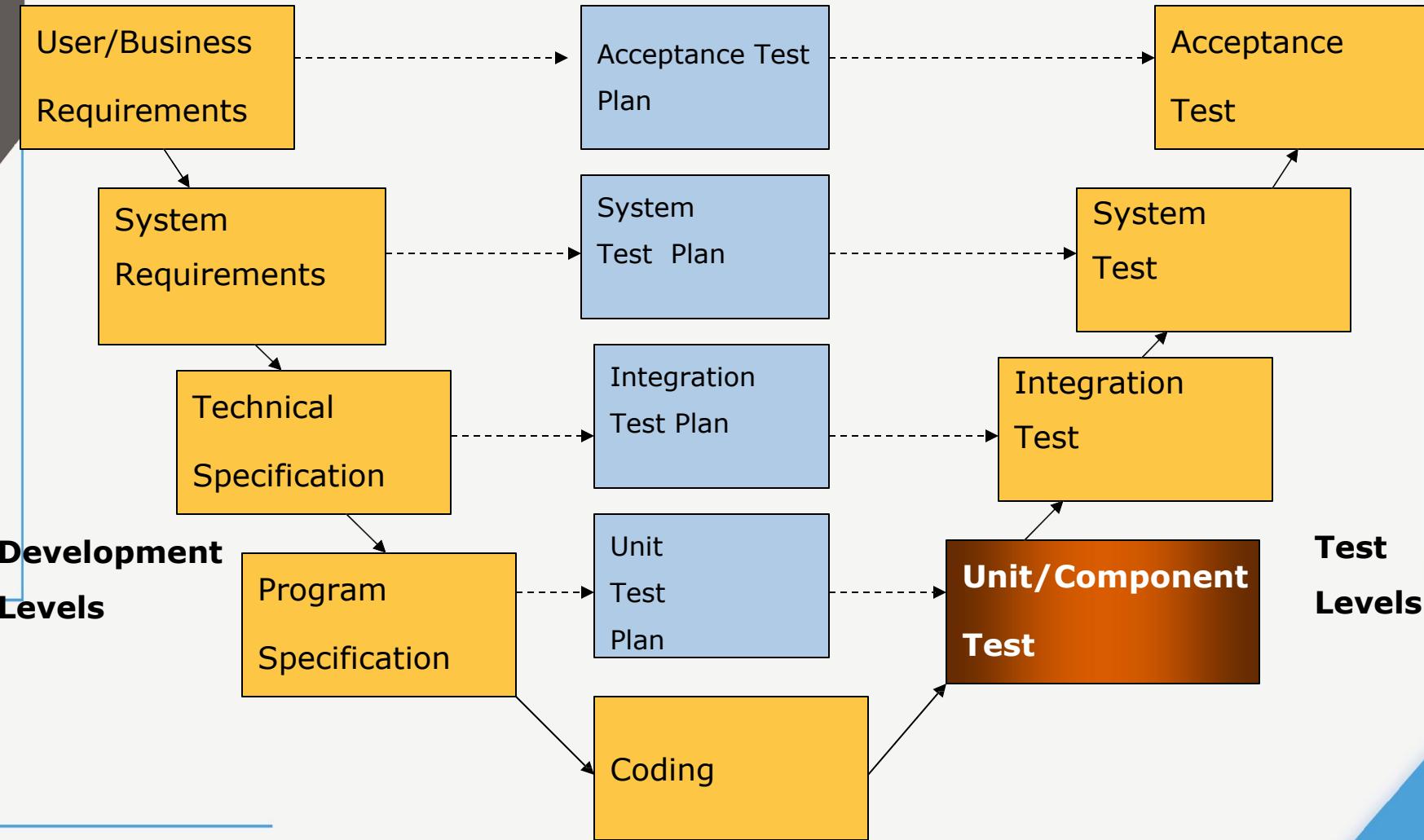
- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.

Software Testing Levels

Agenda

- Component Testing (Unit Testing)
- Integration Testing
 - Component Integration Testing
 - System Integration Testing
- System testing
- User Acceptance Testing (UAT)

Component Testing



Component Testing

Component(Unit) – A minimal software item that can be tested in isolation. It means “A unit is the smallest testable part of software.”

Component Testing – The testing of individual software components.

Unit Testing is a level of the software testing process where **individual units/components of a software/system** are tested. The purpose is to validate that each unit of the software performs as designed.

Unit testing is the first level of testing and is performed prior to Integration Testing.

Sometimes known as **Unit Testing, Module Testing or Program Testing**

Component can be tested in isolation – stubs/drivers may be employed

Unit testing **frameworks, drivers, stubs and mock or fake objects** are used to assist in unit testing.

Test cases derived from component specification (module/program spec)

Functional and Non-Functional testing

Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended with debugging

Component Testing

- It usually has one or a few inputs and usually a single output. In procedural programming a unit may be an individual program, function, procedure, etc.
- In object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.)
- Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.
- A unit is the smallest testable part of an application like functions/procedures, classes, interfaces.
- The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.
- A unit test provides a strict, written contract that the piece of code must satisfy.
As a result, it affords several benefits.
- Unit tests find problems early in the development cycle.
- **Unit testing is performed by using the White Box Testing method.**

Component Testing

- Quick and informal defect fixing
 - Test-First/Test-Driven approach – create the tests to drive the design and code construction!
 - Instead of creating a design to tell you how to structure your code, you create a test that defines how a small part of the system should function.
- Three steps:
1. Design test that defines how you think a small part of the software should behave (Incremental development).
 2. Make the test run as easily and quickly as you can. Don't be concerned about the design of code, just get it to work!
 3. Clean up the code. Now that the code is working correctly, take a step back and re-factor to remove any duplication or any other problems that were introduced to get the test to run.

Component Testing

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

Component Testing

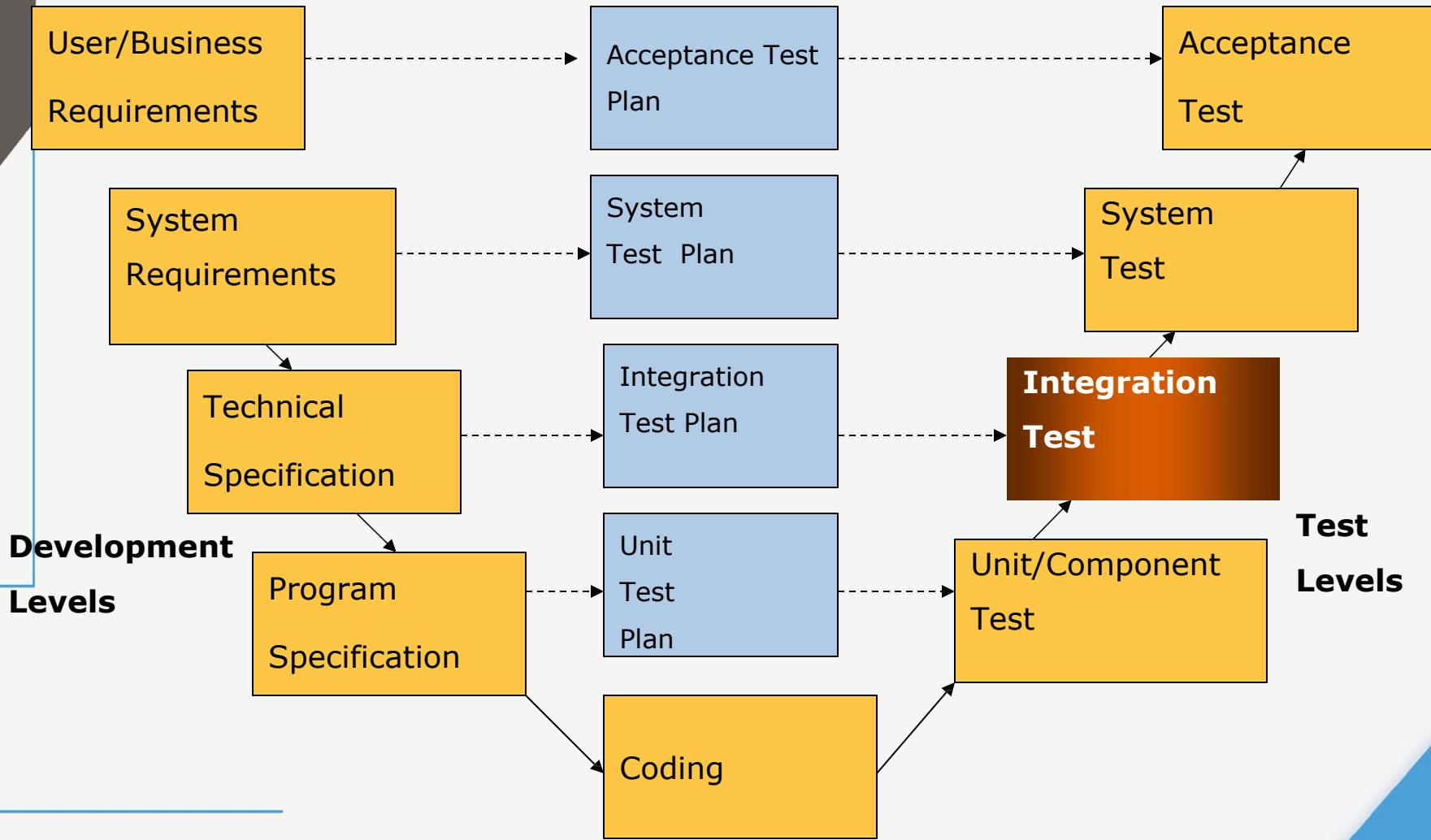
Limitations of tests

- Unit testing can't be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
- Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

Building Unit Test Cases

- Unit testing is commonly automated, but may still be performed manually. The IEEE does not favor one over the other. A manual approach to unit testing may employ a step-by-step instructional document.
- Under the automated approach-
 - A developer could write another section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is done.
 - They could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and pasting the function to its own testing environment to other than its natural environment. Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product. These dependencies can then be eliminated.

Integration Testing



Integration Testing

Integration Testing - Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems

Integration Testing is a level of the software testing process where individual units are combined and tested as a group.

The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.

Integration testing is done by a specific integration tester or test team.

Components may be code modules, operating systems, hardware and even complete systems

There are 2 levels of Integration Testing

- **Component Integration Testing**

- **System Integration Testing**

Need of Integration Testing

A Module in general is designed by an individual software developer who understanding and programming logic may differ from other programmers. Integration testing becomes necessary to verify the software modules work in unity

At the time of module development, there wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence integration testing becomes necessary.

Interfaces of the software modules with the database could be erroneous

External Hardware interfaces, if any, could be erroneous

Inadequate exception handling could cause issues.

Component Integration Testing

Component Integration Testing: Testing performed to expose defects in the interfaces and interaction between integrated components

- Usually formal (records of test design and execution are kept)

- All individual components should be integration tested prior to system testing

- It tests the interactions between software components and is done after component testing.

- The software components themselves may be specified at different times by different specification groups, yet the integration of all of the pieces must work together.

- It is important to cover negative cases as well because components might make assumption with respect to the data.

- The following testing techniques are appropriate for Integration Testing:

- Functional Testing using** Black Box Testing techniques against the interfacing requirements for the component under test

- Non-functional Testing** (where appropriate, for *performance* or *reliability testing* of the component interfaces, for example)

System Integration Testing

- It tests the interactions between different systems and may be done after system testing.
- It verifies the proper execution of software components and proper interfacing between components within the solution.
- The objective of SIT Testing is to validate that all software module dependencies are functionally correct and that data integrity is maintained between separate modules for the entire solution.
- As testing for dependencies between different components is a primary function of SIT Testing, this area is often most subject to Regression Testing.

Integration Testing Methods

During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. For example, whether the cap fits into the body or not.

Any of Black Box Testing, White Box Testing, and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.

There are two types of methods of Integration Testing:

- Bing Bang Integration Testing
- Incremental Integration Testing
 - Top Down Approach
 - Bottom Up Approach

When is Integration Testing performed?

● Integration Testing is performed after Unit Testing and before System Testing.

Who performs Integration Testing?

● Either Developers themselves or independent Testers perform Integration Testing.

Big Bang Integration Testing

In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.

Big Bang testing has the advantage that everything is finished before integration testing starts.

The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.

- Here all components are integrated together at **once**, and then tested.

Big Bang Integration Testing

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces links to be tested could be missed easily.
- Since the integration testing can commence only after “all” the modules are designed, testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Testing

The incremental approach has the advantage that the defects are found early in a smaller assembly when it is relatively easy to detect the cause.

A disadvantage is that it can be time-consuming since stubs and drivers have to be developed and used in the test.

In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. Process continues until all of the modules are joined and tested successfully.

This process is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

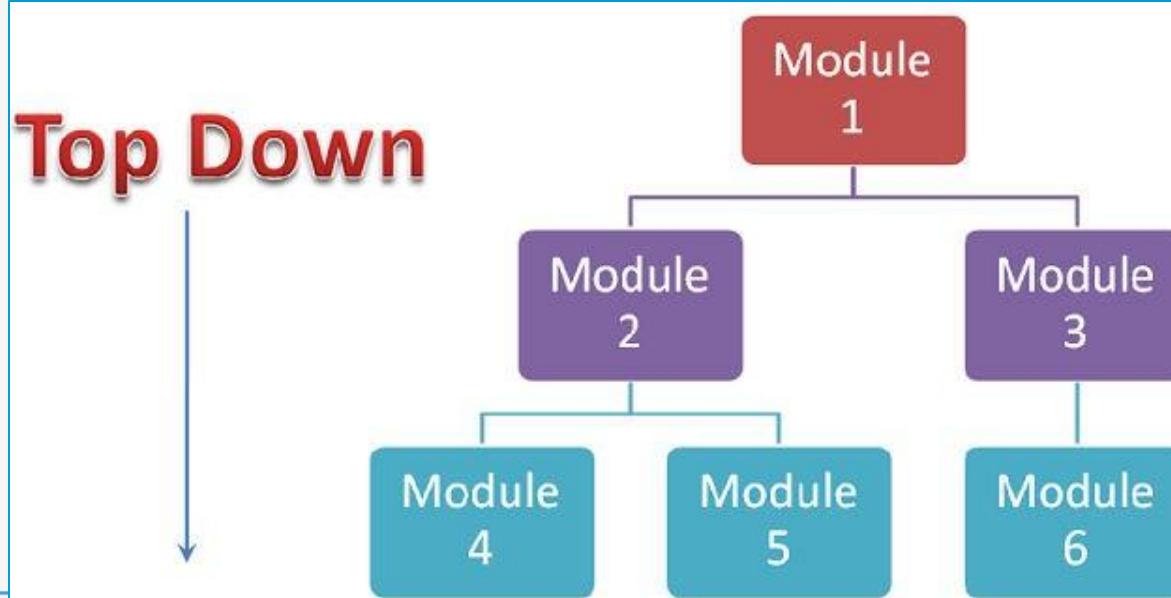
- **Stub:** Is called by the Module under Test.
- **Driver:** Calls the Module to be tested.

Top Down Approach

Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs.

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.



Top Down Approach

Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

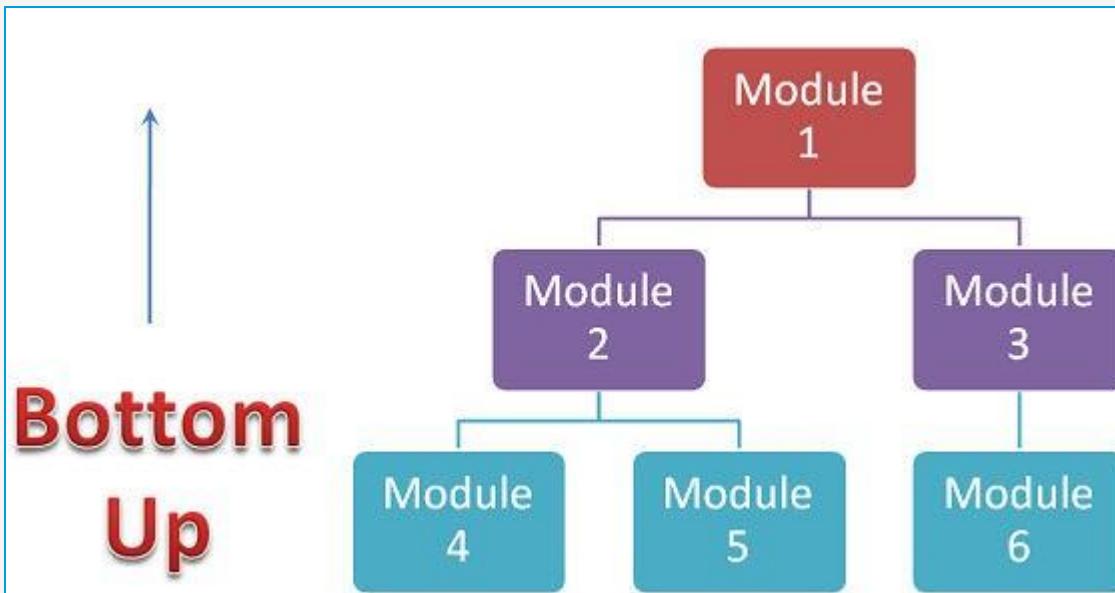
Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

Bottom Up Approach

Testing takes place from the bottom of the control flow upwards.
Components or systems are substituted by drivers.

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing



Bottom Up Approach

Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- Early prototype is not possible

Entry and Exit Criteria

Entry Criteria:

- Unit Tested Components/Modules
- All High prioritized bugs fixed and closed
- All Modules to be code completed and integrated successfully.
- Integration test Plan, test case, scenarios to be signed off and documented.
- Required Test Environment to be set up for Integration testing

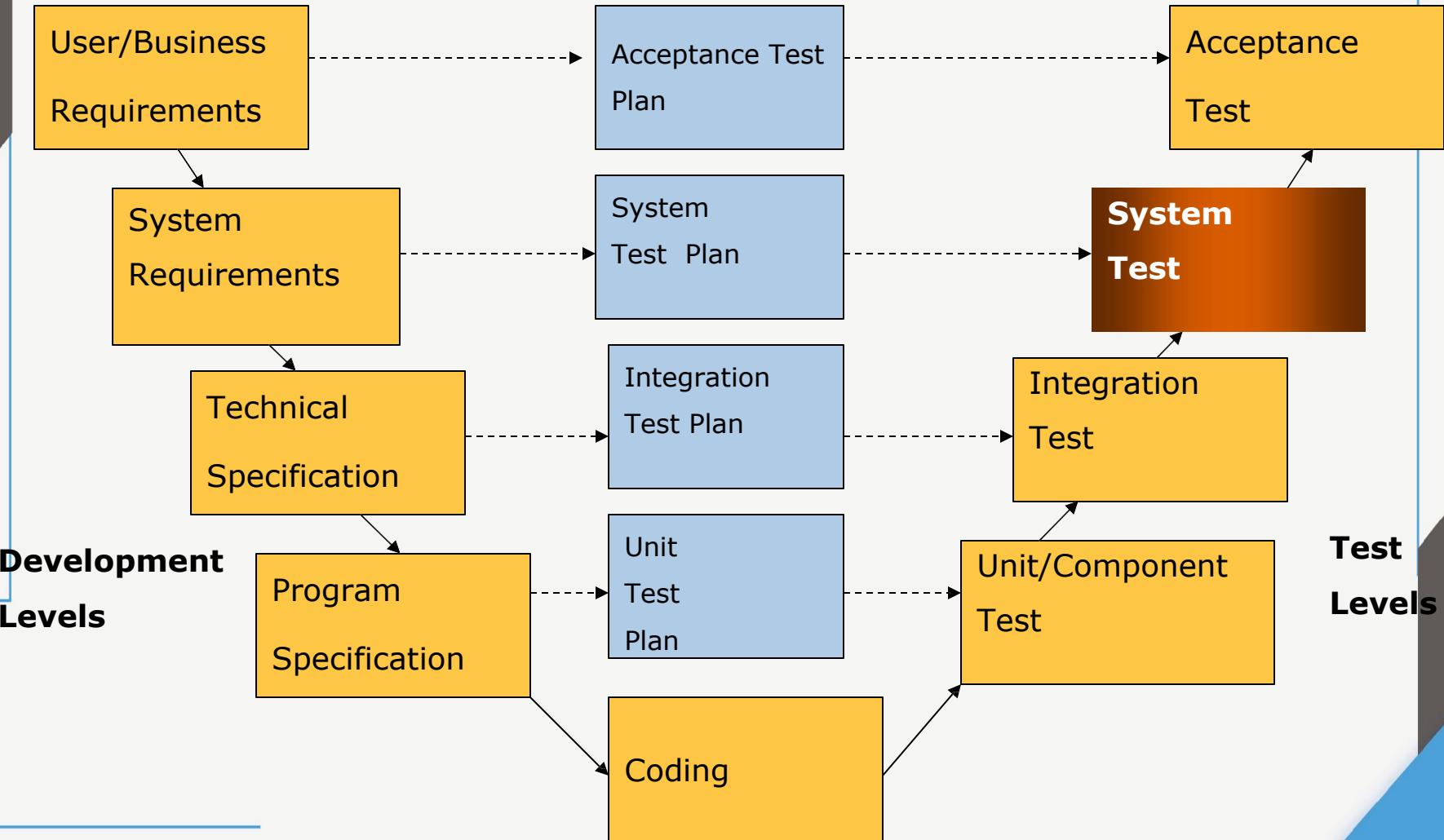
Exit Criteria:

- Successful Testing of Integrated Application.
- Executed Test Cases are documented
- All High prioritized bugs fixed and closed
- Technical documents to be submitted followed by release Notes.

Limitations

- Any condition not specified in integration tests, apart from the confirmation of the execution of the design items is usually not tested.

System Testing



System Testing

- **System Testing - process of testing an integrated system to verify that it meets specified requirements**

- In system testing the behavior of whole system/product is tested as defined by the scope of the development project or product.

- It may include tests based on risks and/or requirement specifications, business process, use cases, or other high level descriptions of system behavior, interactions with the operating systems, and system resources.

- System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose.

- System testing is carried out by **specialists testers or independent testers**.

- System testing should investigate both functional and non-functional requirements of the testing.

- There are two types of System Testing which are:

- **Functional System Testing**
- **Non-Functional System Testing**

System Testing

When is it performed?

- System Testing is performed after Integration Testing and before Acceptance Testing.

Who performs it?

- Normally, independent Testers perform System Testing.

What do you verify in System Testing?

System testing involves testing the software code for following

- **Testing the fully integrated applications** including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End scenario testing..
- Verify thorough **testing of every input** in the application to check for desired outputs.
- Testing of the **user's experience** with the application.

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

Functional System Testing

Functional System Testing : A requirement that specifies a function that a system or system component must perform

A Requirement may exist as a text document and/or a model

There are two types of techniques

- Requirement Based Functional Testing
- Process Based Testing

Functional System Testing Functionality As below:

Accuracy	Provision of right or agreed results or effects
Interoperability	Ability to interact with specified systems
Compliance	Adhere to applicable standards, conventions, regulations or laws
Auditability	Ability to provide adequate and accurate audit data
Suitability	Presence and appropriateness of functions for specified tasks

Requirement Based Testing

- Testing against requirements and specifications
- Test procedures and cases derived from:
 - detailed user requirements
 - system requirements functional specification
 - User documentation/instructions
 - high level System design
- Starts by using the most appropriate black-box testing techniques
- May support this with white-box techniques (e.g. menu structures, web page navigation)
- Risk based approach

Business Process Based Testing

Test procedures and cases derived from:

- Expected user profiles
- Business scenarios
- Use cases

Testing should reflect the business environment and processes in which the system will operate.

Therefore, test cases should be based on real business processes.

Non-functional System Testing

Testing of those requirements that do not relate to functionality

Emphasis on non-functional requirements:

- Performance
- Load
- Data volumes
- Storage
- Recovery
- Usability
- Stress
- Security*

* Note that ISTQB treats this as a Functional test. From the syllabus:

- **'Security Testing A type of functional testing, security testing, investigates the functions (e.g. a firewall) relating to detection of threats, such as viruses, from malicious outsiders.'**

Non-functional System Testing

The non-functional aspects of a system are all the attributes other than business functionality, and are as important as the functional aspects. These include:

- the look and feel and ease of use of the system
- how quickly the system performs
- how much the system can do for the user

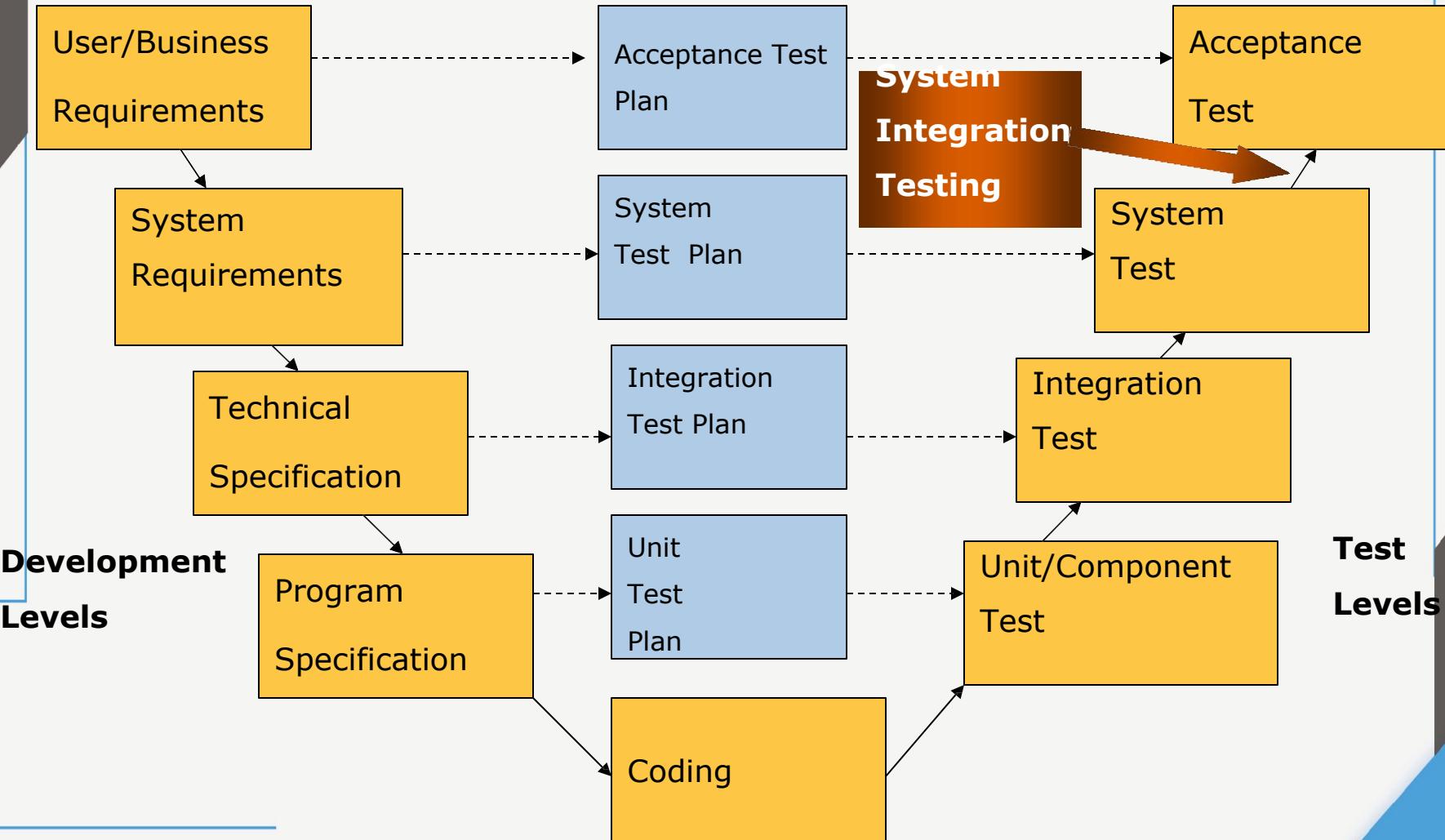
It is also about:

- how easy and quick the system is to install
- how robust it is
- how quickly the system can recover from a crash

Types of System Testing

- **Usability Testing**
- **Load Testing**
- **Regression Testing**
- **Recovery Testing**
- **Migration Testing**
- **Testing Budget**
- **Functional Testing**
- **Hardware/Software Testing**

System Integration Testing



System Integration Testing

System Integration Testing is testing between the ‘System’ and ‘Acceptance’ phases.

The System has already proven to be functionally correct, what remains to be tested is how the system reacts to other systems and/or organisations.

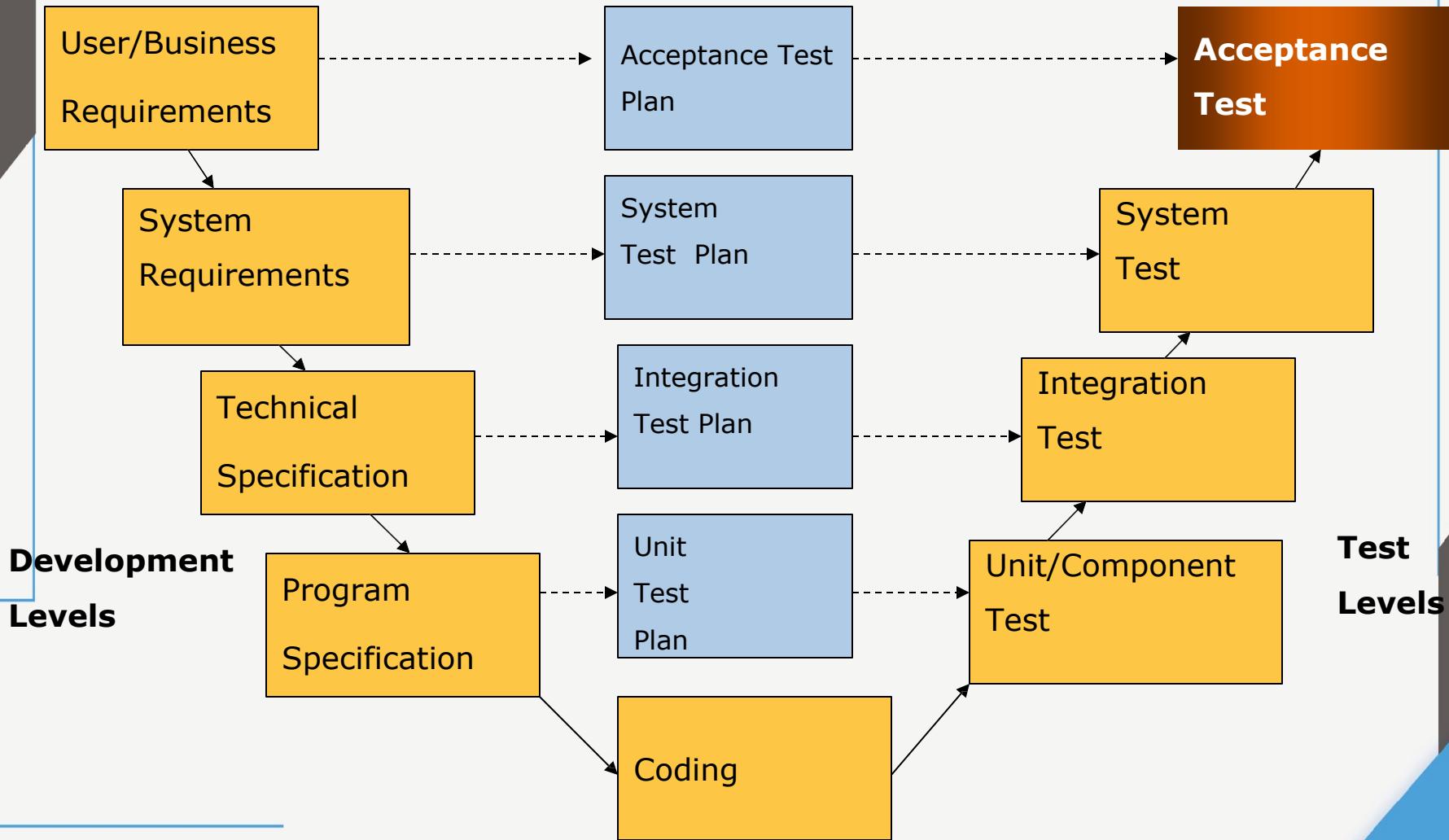
The **objective** of System Integration Testing is to provide confidence that the **system or application** is able to interoperate successfully with other specified software systems and does not have an adverse affect on other systems that may also be present in the **live environment**, or vice versa

System Integration Testing

It is possible that the testing tasks performed during System Integration Testing may be combined with **System Testing**, particularly if the system or application has little or no requirement to interoperate with other systems

- In terms of the *V Model*, Systems Integration Testing corresponds to the Functional and Technical Specification phases of the software development lifecycle
- Having completed Component integration testing and Systems testing, one must execute the plan for system-to-system integration
- Infrastructure may need to be transformed in order to feed to an external system
- Black Box testing techniques used

Acceptance Testing



Acceptance Testing

Acceptance testing: Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

● **Acceptance Testing** is a level of the software testing process where a system is tested for acceptability.

● The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

● After the system test has corrected all or most defects, the system will be delivered to the user or customer for acceptance testing.

● Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well.

● The **goal** of acceptance testing is to **establish confidence in the system**.

● Acceptance testing is most often **focused on a validation type testing**.

● Usually, **Black Box Testing method is used in Acceptance Testing**.

● Testing does not usually follow a strict procedure and is not scripted but is rather **ad-hoc**.

Acceptance Testing

- Usually the responsibility of the **Customer/End user**, though other **stakeholders** may be involved.
- Customer may sub-contract the Acceptance test to a third party
- Goal is to establish confidence in the system/part-system or specific non-functional characteristics (e.g. performance)
- Usually for ensuring the system is ready for deployment into production
- Acceptance testing may occur at more than just a single level, for example:
 - A **Commercial Off the shelf (COTS)** software product may be acceptance tested when it is installed or integrated.
 - Acceptance testing of the usability of the component** may be done during component testing.
 - Acceptance testing of a new functional enhancement** may come before system testing.



Acceptance Testing

When is it performed?

- Acceptance Testing is performed after System Testing and before making the system available for actual use.

Who performs it?

- Internal Acceptance Testing (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.
- External Acceptance Testing is performed by people who are not employees of the organization that developed the software.
 - Customer Acceptance Testing is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software for them. [This is in the case of the software not being owned by the organization that developed it.]
 - User Acceptance Testing (Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers' customers.

Acceptance(Thread) Testing

- Often uses the “Thread Testing” approach:

- A testing technique used to test the business functionality or business logic of the application in an end-to-end manner, in much the same way a User or an operator might interact with the system during its normal use.’*

- This approach is also often used for functional system test

- The same Threads server both test activates

- Often use big bang approach

- Regression testing to ensure changes have not regressed other areas of the system.

Alpha Testing

- It is always performed by the developers at the software development site.
- Sometimes it is also performed by Independent Testing Team.
- Alpha Testing is not open to the market and public
- It is conducted for the software application and project.
- It is always performed in **Virtual Environment**.
- It is always performed within the organization.
- It is the form of Acceptance Testing.
- Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.
- It comes under the category of both White Box Testing and Black Box Testing.

Alpha Testing

- During this phase, the following will be tested in the application:
 - Spelling Mistakes
 - Broken Links
 - Cloudy Directions
- Alpha Testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.
- It is always performed at the developer's premises in the absence of the users.
- It is considered as the User Acceptance Testing (UAT) which is done at developer's area.
- Unit testing, integration testing and system testing when combined are known as alpha testing.

Beta Testing(Field Testing)

- It is always performed by the customers at their own site.
- It is not performed by Independent Testing Team.
- Beta Testing is always open to the market and public.
- It is usually conducted for software product.
- It is performed in **Real Time Environment**.
- It is always performed outside the organization.
- It is also the form of Acceptance Testing.
- Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
- It is only a kind of Black Box Testing.

Beta Testing(Field Testing)

- Beta Testing is always performed at the time when software product and project are marketed.
- It is always performed at the user's premises in the absence of the development team.
- It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.
- Beta testing can be considered “**pre-release**” testing.
- **Pilot Testing** is testing to product on real world as well as collect data on the use of product in the classroom.

Testing Definitions as per ISTQB

- **Unit testing or Component testing:** The testing of individual software components.
- **Integration testing:** Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also component integration testing, system integration testing.
- **Component integration testing:** Testing performed to expose defects in the interfaces and interaction between integrated components.
- **System integration testing:** Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet).
- **System testing:** The process of testing an integrated system to verify that it meets specified requirements.
- **Acceptance testing:** Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

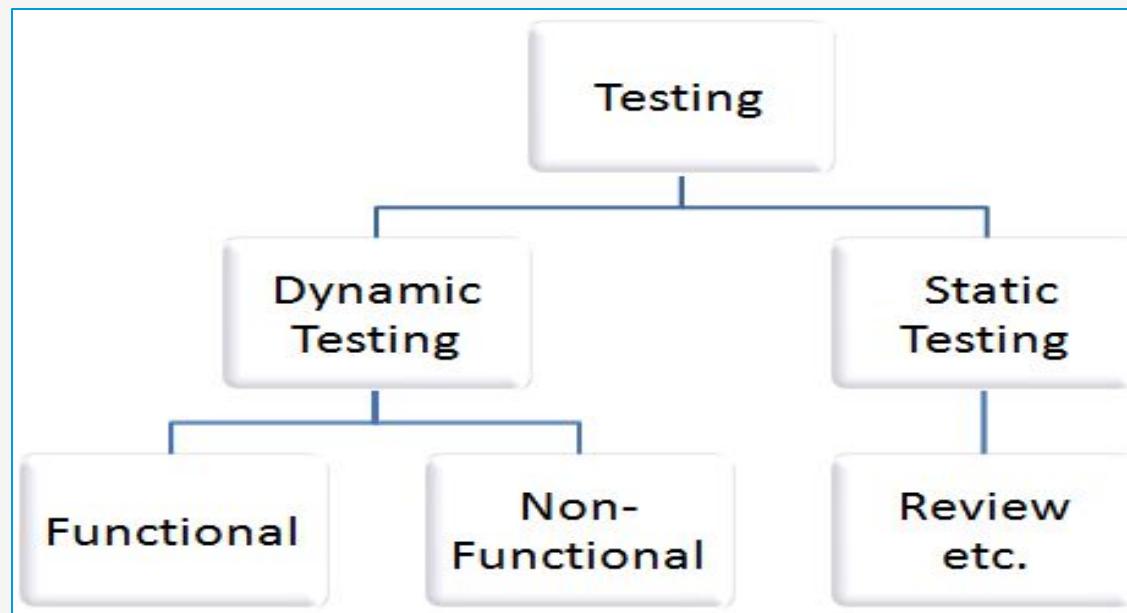
Test Design Techniques

Introduction

Dynamic technique

- Specification-based (black-box, also known as behavioral techniques)
- Structure-based (white-box or structural techniques)
- Experience- based

Static technique



Static vs Dynamic Testing

Dynamic Testing

- This testing technique needs computer for testing.
- It is done during Validation process.
- The software is tested by executing it on computer. Ex: Unit testing, integration testing, and system testing.

Static Testing

- Static testing is the testing of the software work products manually, or with a set of tools, but they are **not executed**.
- It starts early in the Life cycle and so it is done during the verification process.
- **It does not need computer as the testing of program is done without executing the program.** For example: reviewing, walk through, inspection, etc.
- Most static testing techniques can be used to ‘test’ any form of document including source code, design documents and models, functional specifications and requirement specifications.

Static Testing

Under **Static Testing** code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Hence, the name “static”.

Main objective of this testing is to improve the quality of software products by finding errors in early stages of the development cycle. This testing is also called as Non-execution technique or verification testing.

Static testing involves manual or automated reviews of the documents. This review is done during initial phase of testing to catch defect early in STLC. It examines work documents and provides review comments

Static Testing

Work document can be of following:

- Requirement specifications
- Design document
- Source Code
- Test Plans
- Test Cases
- Test Scripts
- Help or User document
- Web Page content

Techniques for Static Testing

Informal Reviews: This is one of the types of review which doesn't follow any process to find errors in the document. Under this technique, you just review the document and give informal comments on it.

Technical Reviews: A team consisting of your peers review the technical specification of the software product and checks whether it is suitable for the project. They try to find any discrepancies in the specifications and standards followed. This review concentrates mainly on the technical document related to the software such as Test Strategy, Test Plan and requirement specification documents.

Walkthrough: The author of the work product explains the product to his team. Participants can ask questions if any. Meeting is led by the author. Scribe makes note of review comments

Techniques for Static Testing

Inspection: The main purpose is to find defects and meeting is led by trained moderator. This review is a formal type of review where it follows strict process to find the defects. Reviewers have checklist to review the work products .They record the defect and inform the participants to rectify those errors.

Static code Review: This is systematic review of the software source code without executing the code. It checks the syntax of the code, coding standards, code optimization, etc. This is also termed as white box testing .This review can be done at any point during development.

Dynamic Testing

Under **Dynamic Testing** code is executed. It checks for functional behavior of software system , memory/CPU usage and overall performance of the system. Hence the name “Dynamic”

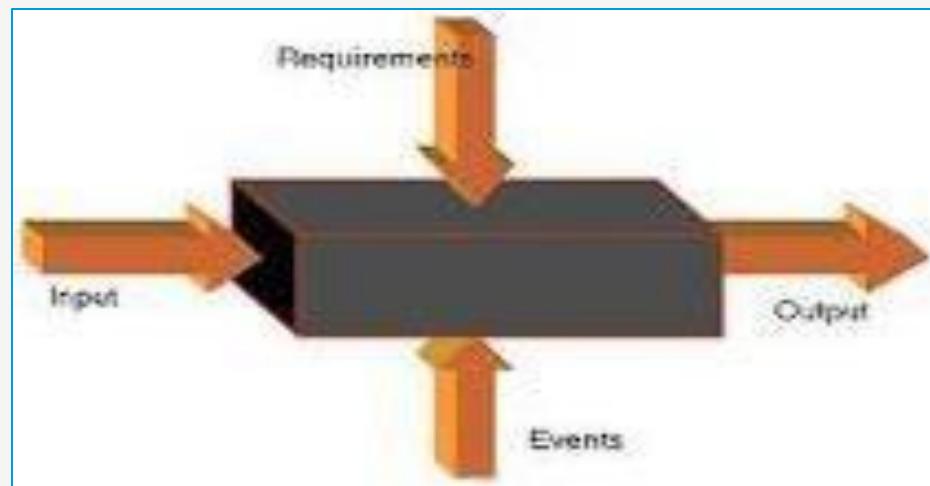
- Main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called as Execution technique or validation testing.
- Dynamic testing executes the software and validates the output with the expected outcome. Dynamic testing is performed at all levels of testing and it can be either black or white box testing.

Techniques for Dynamic Testing

Unit Testing: Under unit testing, individual units or modules are tested by the developers. It involves testing of source code by developers.

Integration Testing: Individual modules are grouped together and tested by the developers. The purpose is to determine that modules are working as expected once they are integrated.

System Testing: System testing is performed on the whole system by checking whether the system or application meets the requirement specification document.



Difference Btn Static & Dynamic

Static Testing	Dynamic Testing
Testing done without executing the program	Testing done by executing the program
This testing does verification process	Dynamic testing does validation process
Static testing is about prevention of defects	Dynamic testing is about finding and fixing the defects
Static testing gives assessment of code and documentation	Dynamic testing gives bugs/bottlenecks in the software system.
Static testing involves checklist and process to be followed	Dynamic testing involves test cases for execution
This testing can be performed before compilation	Dynamic testing is performed after compilation
Static testing covers the structural and statement coverage testing	Dynamic testing covers the executable file of the code
Cost of finding defects and fixing is less	Cost of finding and fixing defects is high
Return on investment will be high as this process involved at early stage	Return on investment will be low as this process involves after the development phase
More reviews comments are highly recommended for good quality	More defects are highly recommended for good quality.

Dynamic Testing Technique

Agenda

- Functional and Non-Functional Testing
- Black Box Testing Techniques
- White Box Testing Techniques
- Experience Based Testing Techniques
- Smoke and Sanity Testing
- Re-Testing and Regression Testing
- End to End Testing
- Positive and Negative Testing
- Maintenance Testing

Functional and Non Functional Testing

Functional Testing

Functional Testing: Testing based on an analysis of the specification of the functionality of a component or system.

'Specification' – E.g. Requirements specification, Use Cases, Functional specification or maybe undocumented.

'Function' – what the system does

Functional test – based on the Functions and features – may be applied at all Test levels (e.g. Component Test, System Test etc.)

Considers the external (not internal) behaviour of the software. Black-Box testing. What it does rather than how it does it. More on this later!

Functional testing verifies that each **function** of the software application operates in conformance with the requirement specification.

Functional Testing

- This testing mainly **involves black box testing** and it is not concerned about the source code of the application.
- Each & every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.
- This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application under Test. The testing can be done either manually or using automation

Functional Testing Examples

- **Web Based Testing :**

- Are you able to login to a system after entering correct credentials?
- Does your payment gateway prompt an error message when you enter incorrect card number?
- Does your “add a customer” screen adds a customer to your records successfully?

- **Desktop Based Testing :**

- Verifies Installation testing, Check for broken lines,
- Testing with different client accounts, Theme change and Print,
- Check for broken links. Warning messages. Resolution change effect on the application

- **Mobile Based Testing :**

- To validate whether the application works as per as requirement whenever the application starts/stops
- To validate whether the application goes into minimized mode whenever there is an incoming phone call. In order to validate the same we need to use a second phone, to call the device

- **Game Based Testing :**

- Takes more time to execute as testers look for game play issues, graphics issues, audio-visual issues, etc.
- Validates whether installation goes smoothly, the app works in minimized mode, the app allows social networking options, supports payment gateways, and many more.

Non-Functional Testing

Non-Functional Testing: Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, interoperability, maintainability and portability

- May be performed at all Test levels (not just Non Functional Systems Testing)
- Measuring the **characteristics** of the system/software that can be quantified on a varying scale- e.g. performance test scaling
- Non-functional testing includes, but is not limited to, performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing and portability testing.

Non-Functional Testing

- It is the testing of “**how**” the system works. Non-functional testing may be performed at all test levels.

- The term non-functional testing describes the tests required to measure characteristics of systems and software that can be quantified on a varying scale, such as response times for performance testing.

- To address this issue, **performance testing is carried out to check & fine tune system response times**. The goal of performance testing is to reduce response time to an acceptable level

- Hence **load testing is carried out to check systems performance at different loads i.e. number of users accessing the system**

Non - Functional Testing Examples

- **Web Based Testing :**
 - Identify the software processes that directly influence the overall performance of the system.
 - **In website number of user/customer will increase , how the website will handled to every customer/user.**
- **Desktop Based Testing :**
 - Numerous other such GUI test cases, the desktop application tester must view
 - Guarantee that error messages are instructive and helpful for the client
 - Memory, and different other issues
- **Mobile Based Testing :**
 - In mobile , automatically will switch off without any reason.
 - To stop the application which is not in our hand.
- **Game Based Testing :**
 - Confirms workability and stability of the software.
 - Validate whether the user interface of the app is as per the screen size of the device and ensure high quality

Functional vs Non-Functional

Functional Testing	Non-Functional Testing
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.	Non-functional testing checks the performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Non functional testing should be performed after functional testing
Manual testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed , scalability are inputs to non-functional testing.
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do manual testing	Tough to do manual testing
Types of Functional testing are <ul style="list-style-type: none"> • Unit Testing • Smoke Testing • Sanity Testing • Integration Testing • White box testing • Black Box testing • User Acceptance testing • Regression Testing 	Types of Nonfunctional testing are <ul style="list-style-type: none"> • Performance Testing • Load Testing • Volume Testing • Stress Testing • Security Testing • Installation Testing • Penetration Testing • Compatibility Testing • Migration Testing

Black box Testing & Technique

Introduction

Black-box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

Specification-based testing technique is also known as ‘black-box’ or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.

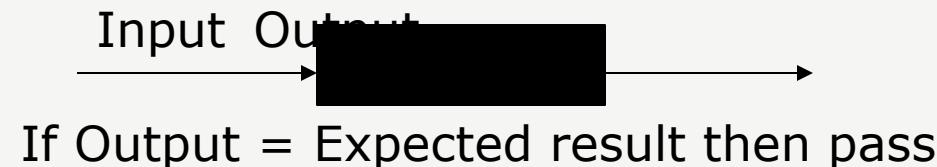
The testers **have no knowledge of how the system or component is structured inside the box**. In black-box testing the tester is concentrating on what the software does, not how it does it.

Specification-based techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists.

- For example, when performing system or acceptance testing, the requirements specification or functional specification may form the basis of the tests.

Introduction(Cont...)

- The technique of testing without having **any knowledge of the interior workings of the application** is Black Box testing.
- What a system does, rather than HOW it does it
- Typically used at System Test phase, although can be useful throughout the test lifecycle
- The tester is oblivious to the **system architecture and does not have access to the source code**.
- Typically, when performing a black box test, **a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon**.



Introduction(Cont...)

Advantages

- Well suited and efficient for large code segments.
- Code Access not required.
- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

Disadvantage

- Limited Coverage since only a selected number of test scenarios are actually performed.
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
- Blind Coverage, since the tester cannot target specific code segments or error prone areas.
- The test cases are difficult to design.

Black Box Testing Examples

Web Based Testing :

- Takes more time to execute as testers look for game play issues, graphic issues, audio-visual issues, etc.
- Validates whether installation goes smoothly, the app works in minimized mode, the app allows social networking options, supports payment gateways, and many more.
- Login by the user is must for accessing the sensitive information.

Desktop Based Testing :

- Resolution change effect on the application
- Installation Testing (Upgrade/Downgrade)

Mobile Based Testing :

- In mobile , automatically will switch off without any reason.
- To stop the application which is not in our hand.

Game Based Testing :

- the game tester must know how to play the game, utilization of the gamepad, know the game flow and the rules.

Techniques of Black Box Testing

- There are four specification-based or black-box technique:

- Equivalence partitioning
- Boundary value analysis
- Decision tables
- State transition testing
- Use-case Testing
- Other Black Box Testing
 - Syntax or Pattern Testing

Equivalence Partitioning(E.P.)

- Aim is to treat groups of inputs as equivalent and to select one representative input to test them all

- EP can be used for all Levels of Testing

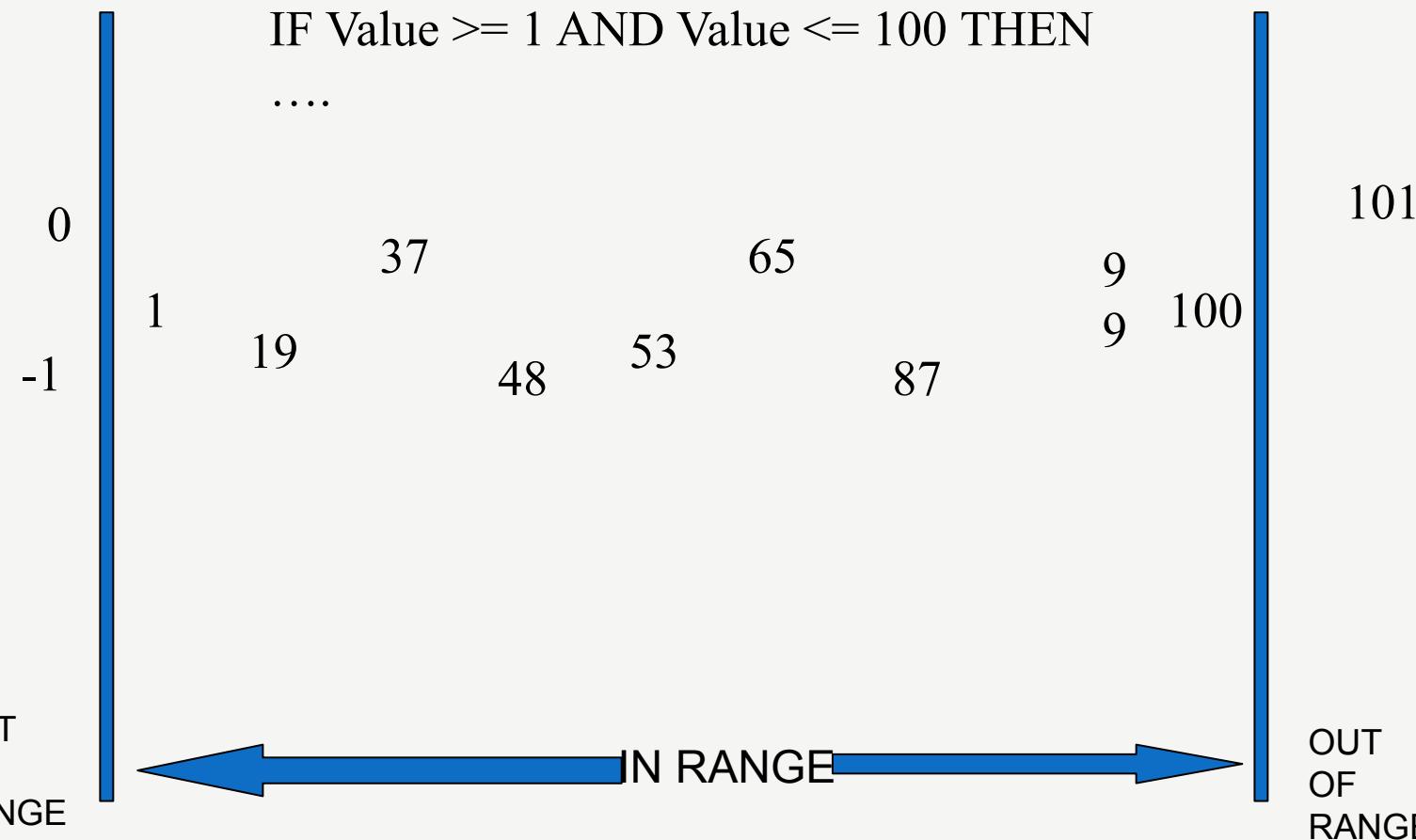
- Equivalence partitioning is the process of defining the optimum number of tests by:

- Reviewing documents such as the Functional Design Specification and Detailed Design Specification, and identifying each input condition within a function,
- Selecting input data that is representative of all other data that would likely invoke the same process for that particular condition.

- If we want to test the following IF statement: “If value is between 1 and 100 (inclusive) (e.g value ≥ 1 and value ≤ 100) Then...”

- We could put a range of numbers as shown in the below figure.

Equivalence Partitioning(E.P.)



Equivalence Partitioning(E.P.)

The numbers fall into a partition where each would have the same, or equivalent, result i.e. an Equivalence Partition (EP) or Equivalence Class

EP says that by testing just one value we have tested the partition (typically a mid-point value is used). It assumes that:

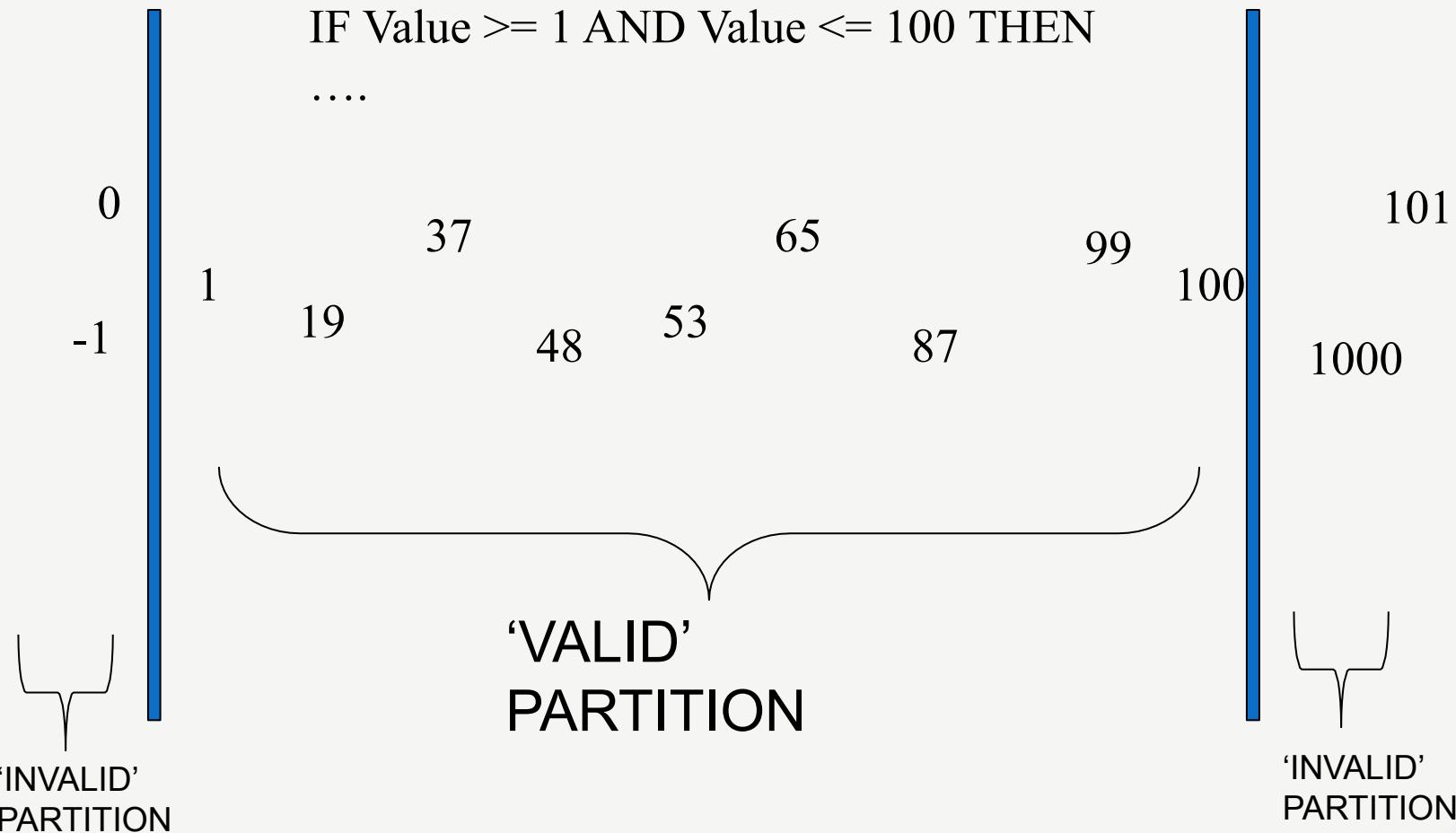
- If one value finds a bug, the others probably will too
- If one doesn't find a bug, the others probably won't either

In EP we must identify Valid Equivalence partitions and Invalid Equivalence partitions where applicable (typically in range tests)

The Valid partition is bounded by the values 1 and 100

Plus there are 2 Invalid partitions

Equivalence Partitioning(E.P.)



Equivalence Partitioning(E.P.)

- Time would be wasted by specifying test cases that covered a range of values within each of the **three partitions**, unless the code was designed in an unusual way

- There are more effective techniques that can be used to find bugs in such circumstances (such as code inspection)

- EP can help reduce the number of tests from a list of all possible inputs to a minimum set that would still test each partition

- If the tester chooses the right partitions, the testing will be accurate and efficient

- EP is used to achieve good input and output coverage, knowing exhaustive testing is often impossible

- It can be applied to human input, input via interfaces to a system, or interface parameters in integration testing

Boundary Value Analysis(B.V.A.)

- Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near **the limits of valid ranges**

- Boundary value analysis is a method which **refines** equivalence partitioning.

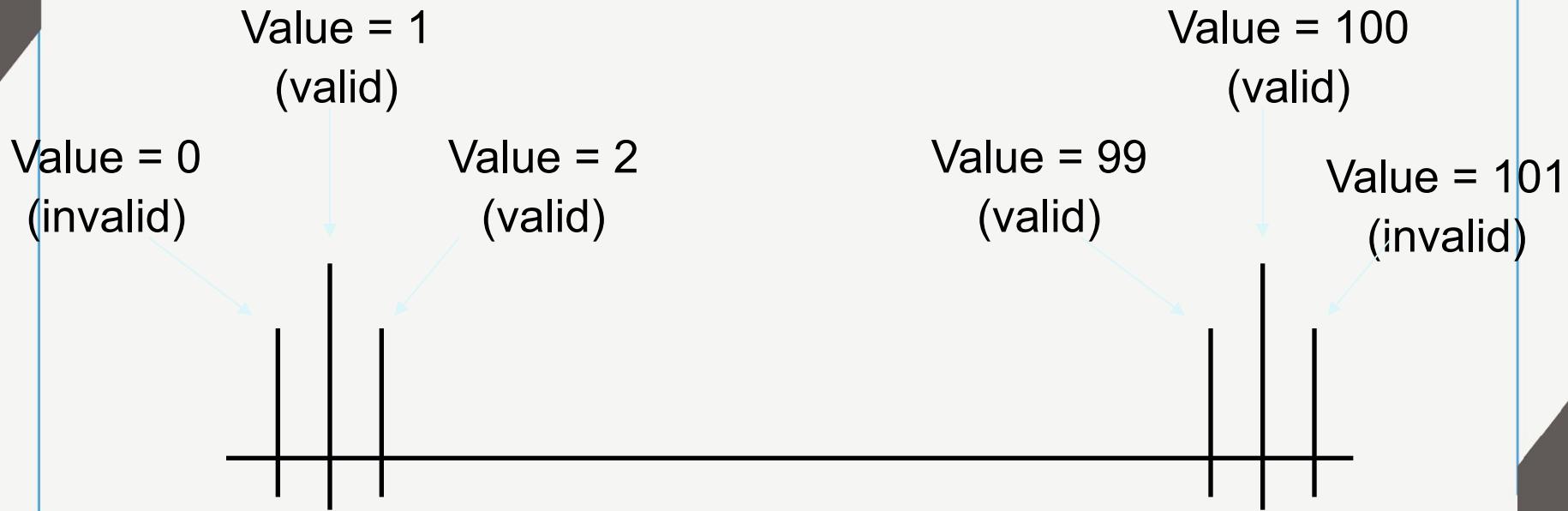
- Boundary value analysis generates test cases that highlight errors better than equivalence partitioning.

- The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes.

- At those points when input values change from valid to invalid errors are most likely to occur.

- Boundary Value Analysis (BVA) uses the same analysis of partitions as EP and is usually used in conjunction with EP in test case design

Boundary Value Analysis(B.V.A.)



Boundary Value Analysis(B.V.A.)

- BVA operates on the basis that experience shows us that errors are most likely to exist **at the boundaries between partitions** and in doing so incorporates a degree of negative testing into the test design

- BVA Test cases are designed to exercise the software on and at either side of boundary values

- Find the boundary and then test one value above and below it

- Always results in **two test cases per boundary for valid inputs and three tests cases per boundary for all inputs**

- inputs should be in the smallest significant values for the boundary (e.g. Boundary of 'a > 10.0' should result in test values of 10.0, 10.1 & 10.2) only applicable for numeric (and date) fields

Decision Table

- The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs.
- However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface.
- The other two specification-based software testing techniques, decision tables and state transition testing are more focused on business logic or business rules.**
- A decision table is a good way to deal with combinations of things (e.g. inputs).**
- This technique is sometimes also referred to as a '**cause-effect**' table. The reason for this is that there is an associated logic diagramming technique called '**cause-effect graphing**' which was sometimes used to help derive the decision table (Myers describes this as a combinatorial logic network [Myers, 1979]). However, most people find it more useful just to use the table described in [Copeland, 2003].

Decision Table

- Table based technique where

- Inputs to the system are recorded
- Outputs to the system are defined

Inputs are usually defined in terms of actions which are Boolean (true or false)

- Outputs are recorded against each unique combination of inputs

Using the **Decision Table the relationships between the inputs and the possible outputs are mapped together**

- As with State Transition testing, an excellent tool to capture certain types of system requirements and to document internal system design.

- As such can be used for a number of test levels

- Especially useful for complex business rules

Decision Table

- Each column of the table corresponds to a business rule that defines a unique combination of conditions that result in the execution of the actions associated with that rule
- The strength of Decision Table testing is that it creates combinations of conditions that might not otherwise have been exercised during testing

	Test 1	Test 2	Test 3
Inputs / Actions	Input 1	T	T
	Input 2	T	F
	Input 3	T	DON'T CARE
	Input 4	F	T
Output / Response	Response 1	Y	N
	Response 2	Y	Y
	Response 3	N	N

Decision Table

What will be the outcome of the following Scenarios?

Joe is a 22 year old non smoker who goes to the gym 4 times / week and has no history of heart attacks in his family

Kevin is 62 year old non smoker who swims twice a week and plays tennis. He has no history of heart attacks in his family

	Test 1	Test 2	Test 3
> 55 yrs old	F	T	T
Smoker	F	T	F
Exercises 3 times a week +	T	F	T
History of Heart Attacks	F	T	F
Insure	Y	N	Y
Offer 10% Discount	N	N	Y
Offer 30% Discount	Y	N	N

State Transaction Testing

State Transition Testing uses the following terms:

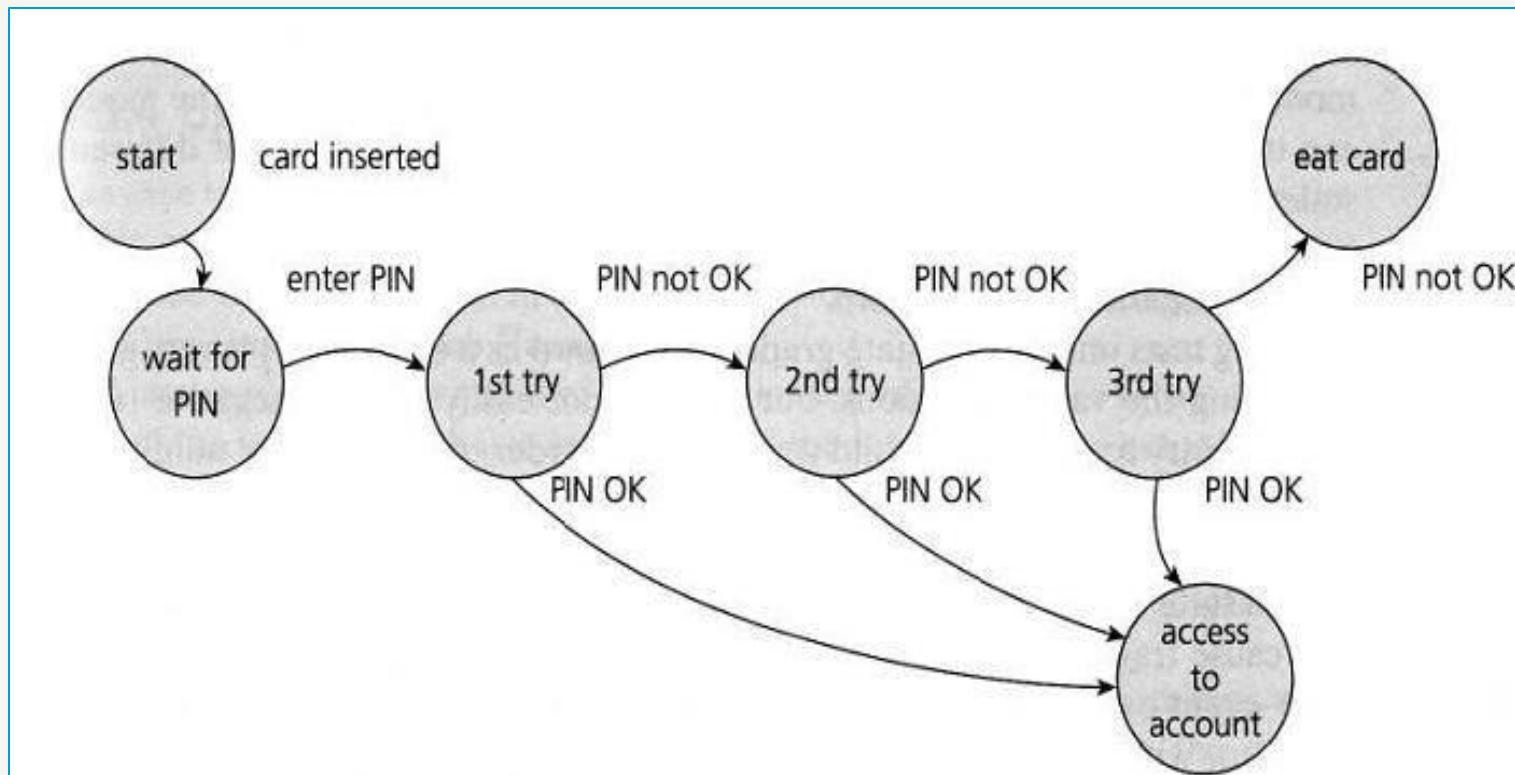
- **State Diagram:** A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another. [IEEE 610]
- **State Table:** A grid showing the resulting transitions for each state combined with each possible event, showing both valid and invalid transitions.
- **State Transition:** A transition between two states of a component or system.
- **State Transition Testing:** A black box test design technique in which test cases are designed to execute valid and invalid state transitions. Also known as N-switch testing.
- An excellent tool to capture certain types of system requirements and to document internal system design. As such can be used for a number of test levels
- Often used in testing:
 - Screen dialogues

State Transaction Testing

State transition testing uses the same principles as the State Transition Diagramming design technique.

- State transition testing is used where some aspect of the system can be described in what is called a '**finite state machine**'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the '**machine**'. This is the model on which the system and the tests are based.
- Any system where you get a **different output for the same input**, depending on what has happened before, **is a finite state system**.
- **A finite state system is often shown as a state diagram.(next slide given example)**
- One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be.
 - Where a part of the system is **more important** (that is, requires more testing) a greater depth of detail can be modeled.
 - Where the system is **less important** (requires less testing), the model can use a single state to signify what would otherwise be a series of different states.

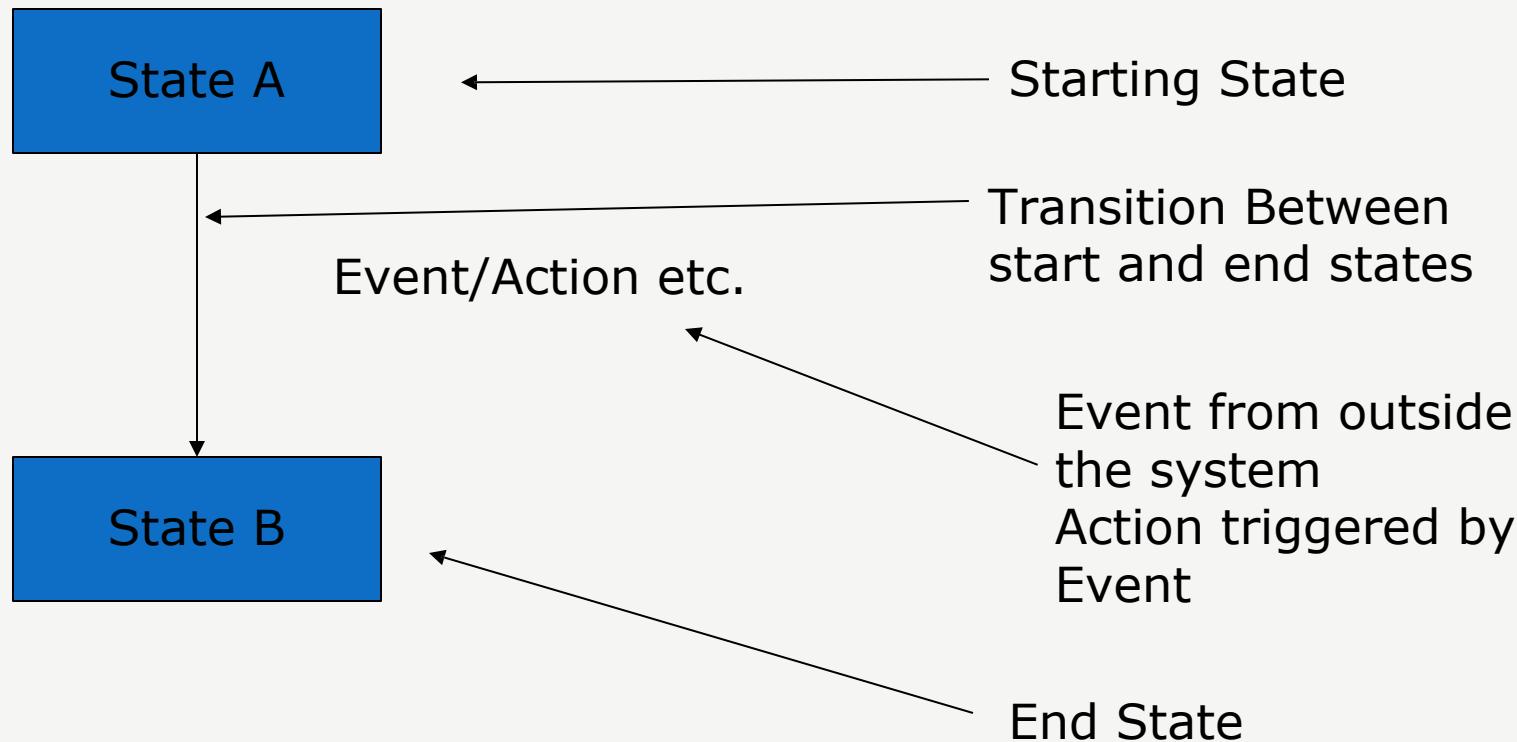
State Transaction Example



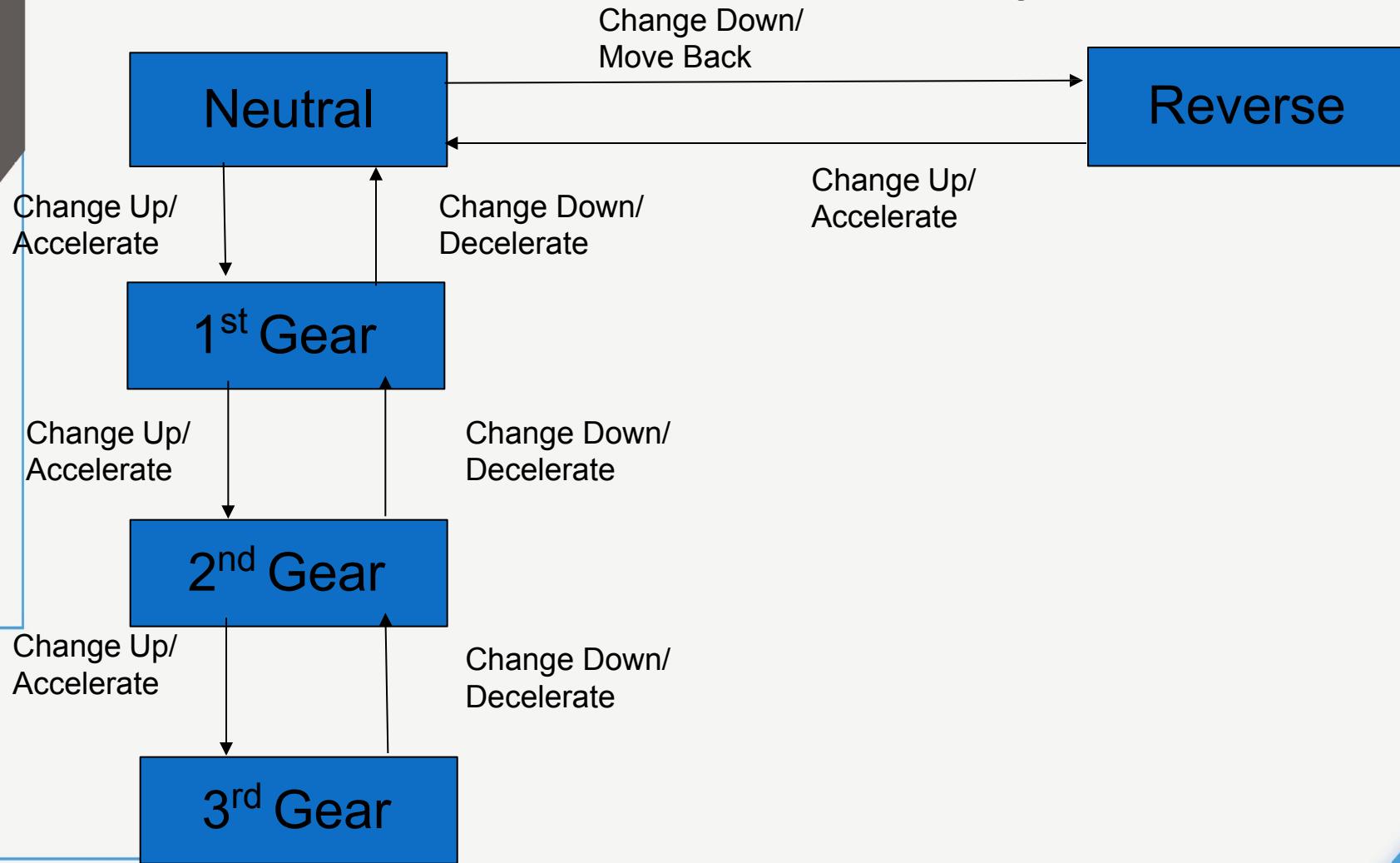
State Transaction Example

- First test case here would be the normal situation, where the correct PIN is entered the first time.
- A second test (to visit every state) would be to enter an incorrect PIN each time, so that the system eats the card.
- A third test we can do where the PIN was incorrect the first time but OK the second time and another test where the PIN was correct on the third try. These tests are probably less important than the first two.
- Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from 'access account' which just goes back to 'access account' for an action such as 'request balance'.

State Transaction Examples



State Transaction Examples



Switch Coverage of State Transaction

- Switch Coverage is a method of determining the number tests based on the number of “hops” between transitions. Some times known as Chow
- These hops can be used to determine the VALID tests

0-Switch Coverage (1 hop)

R□N
N□R
N□1
1□N
1□2
2□1
2□3
3□2

1-Switch Coverage (2 hops)

R□N□1
R□N□R
N□R□N
N□1□2
N□1□N
1□N□R
Etc.

State Table of Transaction

- While Switch testing helps determine the valid tests, we also need to look for the invalid tests.
- Invalid tests are those identified by a null output in this case
 - Changing down from reverse
 - Changing up from 3rd

	Change Up	Change Down
R	Acc / Neutral	Null
N	Acc/1 st Gear	Dec / Reverse
1st	Acc/ 2 nd Gear	Dec / Neutral
2nd	Acc / 3 rd gear	Dec / 1st Gear
3rd	Null	Dec / 2 nd Gear

Other Black Box Techniques

Syntax(Pattern) Testing: test cases are prepared to exercise the rule governing the format of data in a system (e.g. a Zip or Postal Code, a telephone number)

Random Testing: test cases are selected, possibly using a pseudo-random generation algorithm, to match an operational profile

WhiteBox Testing & Technique

Introduction

White Box Testing: *Testing based on an analysis of the internal structure of the component or system.*

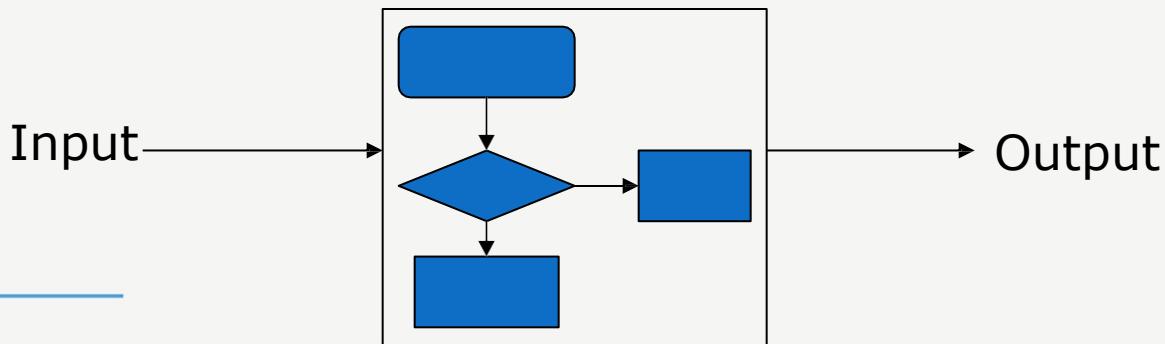
Structure-based testing technique is also known as '**white-box**' or '**glass-box**' testing technique because here **the testers require knowledge of how the software is implemented, how it works.**

In white-box testing the tester is concentrating on how the software does it.

- For example, a structural technique may be concerned with exercising loops in the software.
- Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.
- Structure-based techniques are also used in system and acceptance testing, but the structures are different.
- For example, the coverage of menu options or major business transactions could be the structural element in system or acceptance testing.

Introduction(Cont...)

- Testing based upon the structure of the code
- Typically undertaken at Component and Component Integration Test phases by development teams
- White box testing is the detailed investigation of internal logic and structure of the code.
- White box testing is also called **glass testing or open box testing**. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.



Structural Testing

Structural testing: Testing based on an analysis of the internal structure of the component or system

Also known as **White Box Testing or Glass Box Testing**

May be performed at all Test levels but more commonly during Component Test and Component Integration Test

Coverage measured as a % of items tested – i.e. how much the structure has been tested

May be based on the system Architecture – e.g. a calling hierarchy

Need for use of Test Tools – e.g. for testing coverage of Statements and Decision in the code

More on White Box testing and Coverage later ...

White Box Testing Examples

- **Web Based Testing :**
 - Analyze the logic by reading the code
 - Code optimization Suggest if any optimization can be done in the code which is better than the existing one
 - Deploy the code using different web servers which could be a case study even if the product is not supporting other web servers
- **Desktop Based Testing :**
 - When we debug the code when we writing
- **Mobile Based Testing :**
 - The Android SDK and related plugin for Eclipse
 - Android devices enabled for development and debugging, with appropriate USB drivers as necessary
- **Game Based Testing :**
 - When we connect with remote device , so which device we connect will check in code
 - When some debug the code and play at that time game.

Test/Code Coverage

Test coverage measures the amount of testing performed by a set of test. Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage and is known as test coverage.

The basic coverage measure is where the ‘coverage item’ is whatever we have been able to count and see whether a test has exercised or used this item

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

There is danger in using a coverage measure. But, 100% coverage does *not* mean 100% tested. Coverage techniques measure only one dimension of a multi-dimensional concept. Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other doesn't.

Benefit & Drawback of Test/Code Coverage

BENEFIT:

- It creates additional test cases to increase coverage
- It helps in finding areas of a program not exercised by a set of test cases
- It helps in determining a quantitative measure of code coverage, which indirectly measures the quality of the application or product.

DRAWBACK :

- One drawback of code coverage measurement is that it measures coverage of what has been written, i.e. the code itself; it cannot say anything about the software that has not been written.
- If a specified function has not been implemented or a function was omitted from the specification, then structure-based techniques cannot say anything about them it only looks at a structure which is already there.

Types of Coverage

The different types of coverage
are:

- Statement coverage
- Decision coverage
- Condition coverage

Statement/Segment Coverage

- The statement coverage is also known as line **coverage or segment coverage**.
- The statement coverage **covers only the true conditions**.
- Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
- In this process each and every line of code needs to be checked and executed.
- Aim is to display that all executable statements have been run at least once

The statement coverage can be calculated as shown below:

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

Statement/Segment Coverage

ADVANTAGE:

- It verifies what the written code is expected to do and not to do
- It measures the quality of code written
- It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

DISADVANTAGE:

- It cannot test the false conditions.
- It does not report that whether the loop reaches its termination condition.
- It does not understand the logical operators.

Decision/Branch Coverage

Decision coverage **also known as branch coverage** or all-edges coverage.

It **covers both the true and false conditions** unlikely the statement coverage.

A branch is the outcome of a decision, so branch coverage simply measures which decision outcomes have been tested.

Aim is to demonstrate that all Decisions have been run at least once

Decision and Branch Test coverage for a piece of code is often the same, but not always

With an IF statement, the exit can either be TRUE or FALSE, depending on the value of the logical condition that comes after IF.

The decision coverage can be calculated as shown below:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\%$$

Decision/Branch Coverage

A decision is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL, JUMP, GO TO), or a CASE statement, where there are two or more outcomes from the statement.

ADVANTAGES:

- To validate that all the branches in the code are reached
- To ensure that no branches lead to any abnormality of the program's operation
- It eliminate problems that occur with statement coverage testing

DISADVANTAGES:

- This metric ignores branches within Boolean expressions which occur due to short-circuit operators.

NOTE:

- Branch Coverage Testing \geq Statement Coverage Testing

Condition Coverage

- This is closely related to decision coverage but has better sensitivity to the control flow.

- However, **full condition coverage does not guarantee full decision coverage.**

- Condition coverage reports the true or false outcome of each condition.

- Condition coverage measures the conditions independently of each other.

Statement Coverage Example

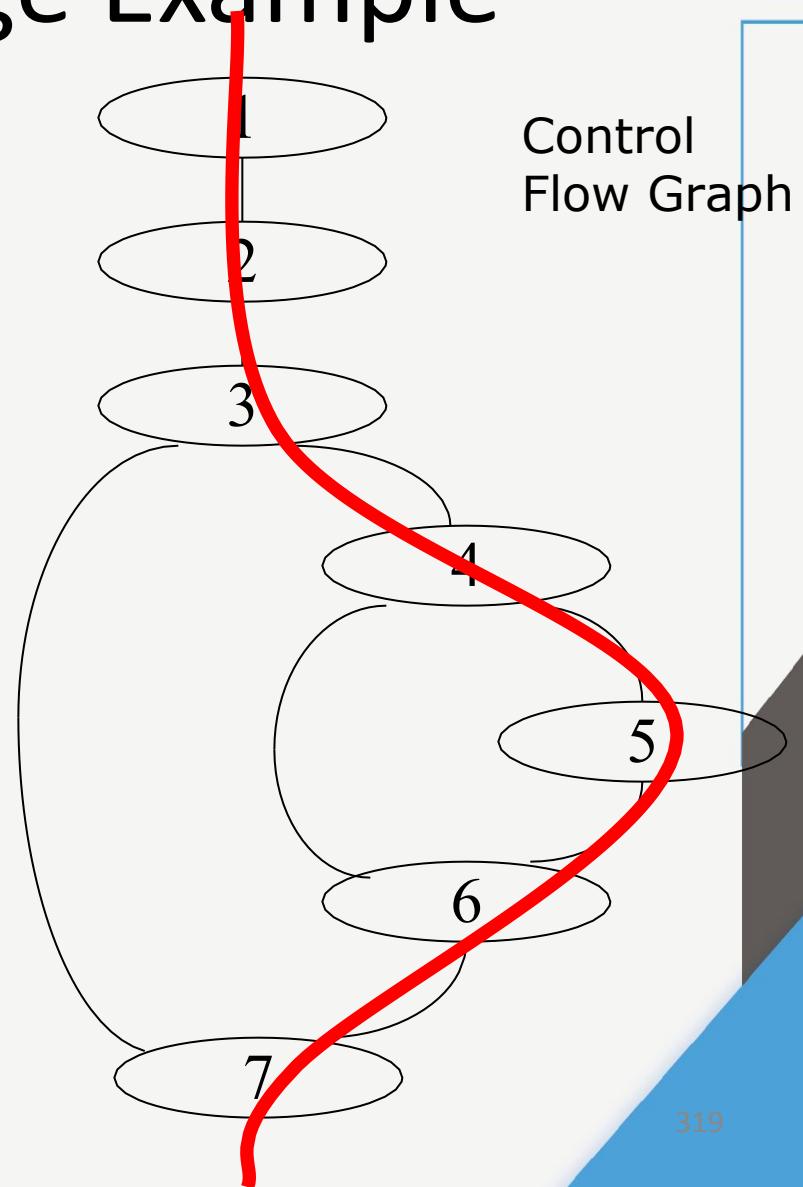
1. Read vehicle
2. Read colour
3. If vehicle = 'Car' Then
4. If colour = 'Red' Then
5. Print "Fast"
6. End If
7. End If

Answer :

- Statement Coverage is 1.

Reason:

- Because of we need to take single input to cover the single statement
- Consider Input is:
- Vehicle = car and color = red



Decision Coverage Example

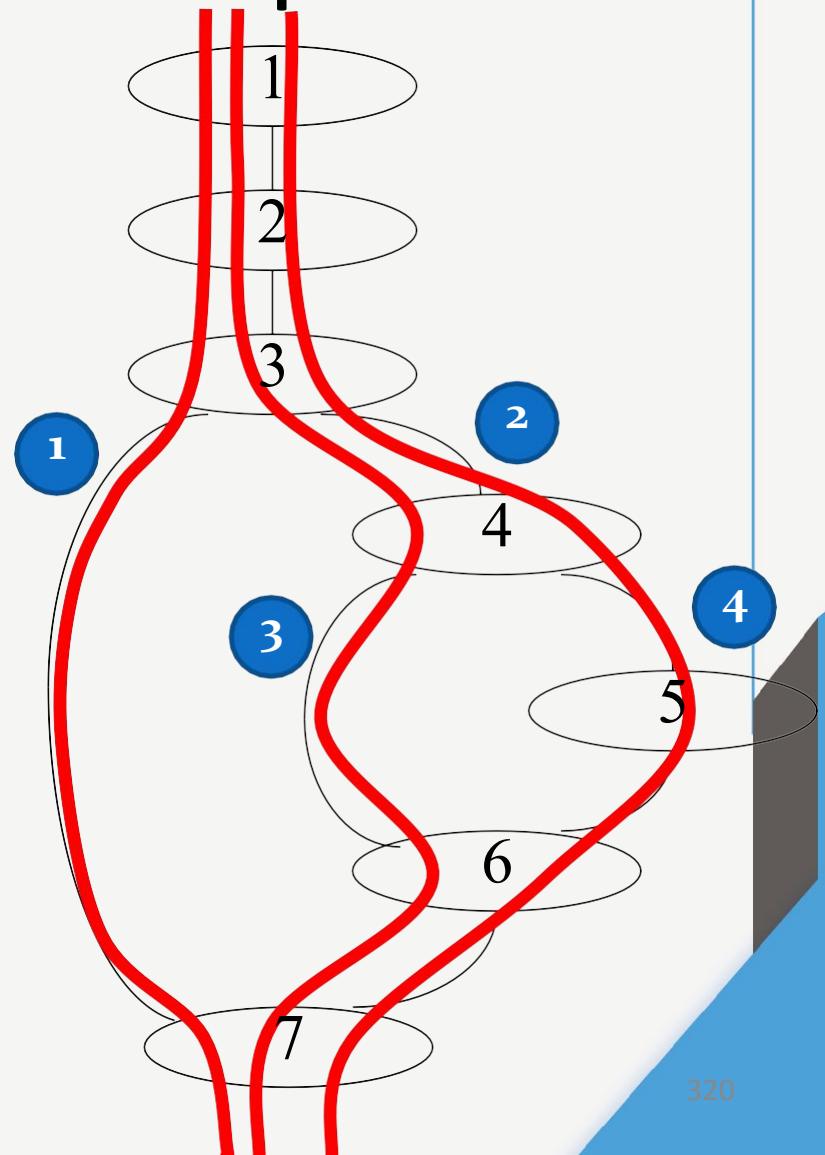
1. Read vehicle
2. Read colour
3. If vehicle = 'Car' Then
4. If colour = 'Red' Then
5. Print "Fast"
6. End If
7. End If

Answer :

- Decision/Branch Coverage is 3.

Reason:

- Because of we need to take three input to cover the four branches
- Consider Input is:
 - (Vehicle = car and color = red) cover branches 2 and 4
 - (Vehicle = scooter and color = black) cover branch 1
 - (Vehicle = car and color = black) cover branch 2 and 3.



Code Cover in Flow Chart

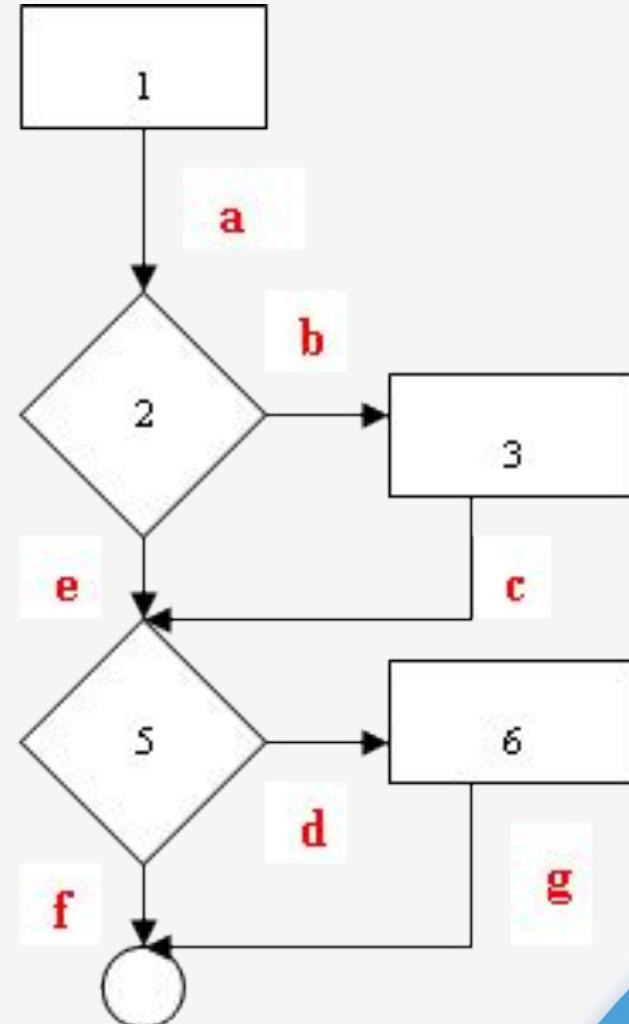
1. Read A
2. If A > 40 Then
 3. $A = A * 2$
4. End If
5. If A > 100 Then
 6. $A = A - 10$
7. End If

Answer :

- Decision/Branch Coverage is 2.

Reason:

- Because of we need to take two input to cover the four branches
- Consider Input is:
 - $A = 60$ cover branch no 2 and 4
 - $A = 10$ cover branch no 1 and 3



Statement Coverage Example

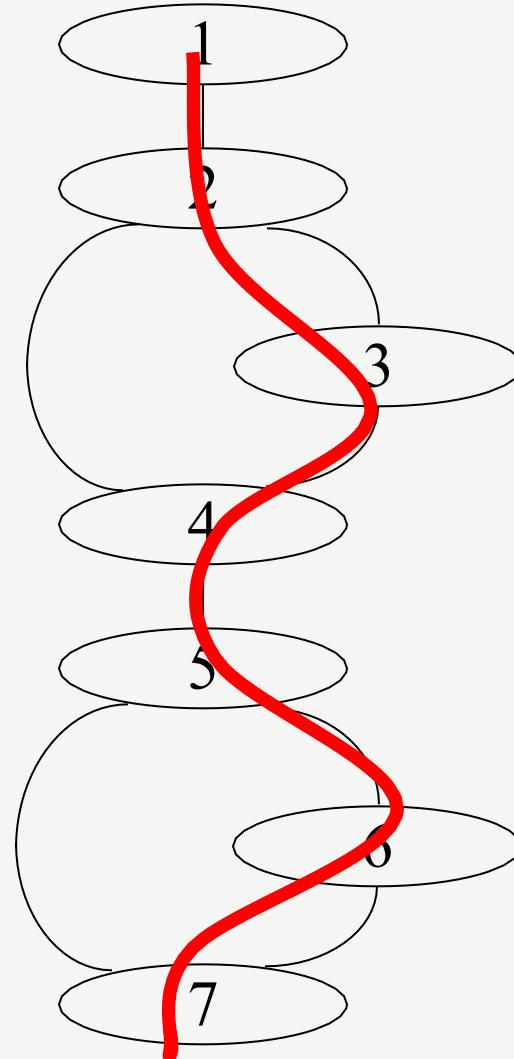
1. Read A
2. If A > 40 Then
3. A = A * 2
4. End If
5. If A > 100 Then
6. A = A – 10
7. End If

Answer :

- Statement Coverage is 1.

Reason:

- Because of we need to take single input to cover the two statement
- Consider Input is:
- A = 60



Decision Coverage Example

1. Read A
2. If $A > 40$ Then
3. $A = A * 2$
4. End If
5. If $A > 100$ Then
6. $A = A - 10$
7. End If

Answer :

- Decision/Branch Coverage is 2.

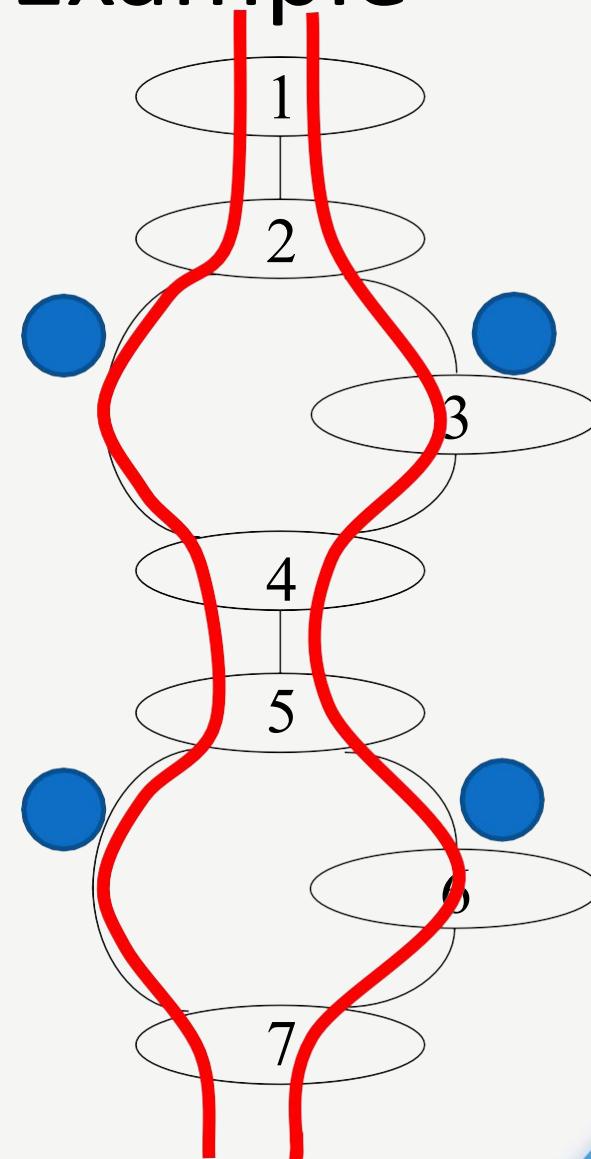
Reason:

- Because of we need to take two input to cover the four branches

- Consider Input is:

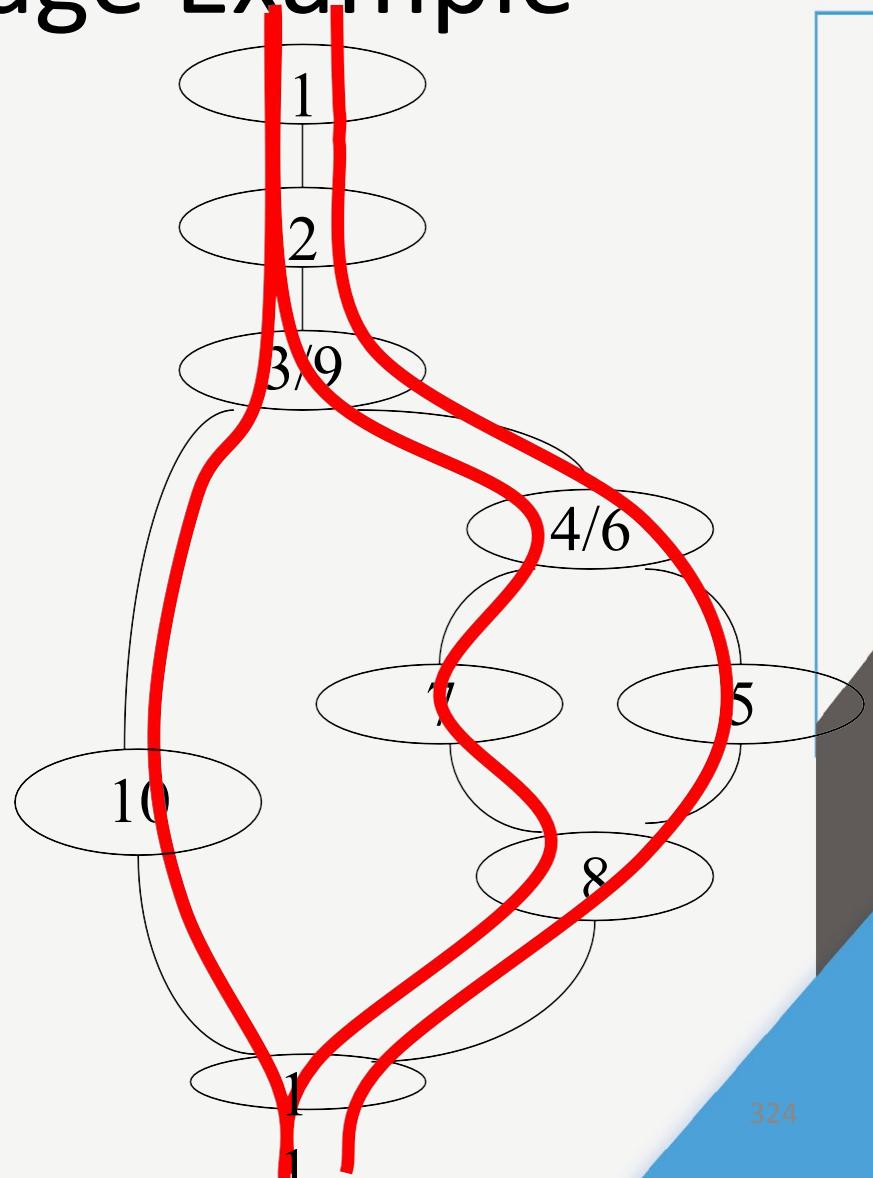
- $A = 60$ cover branch no 2 and 4

- $A = 10$ cover branch no 1 and 3



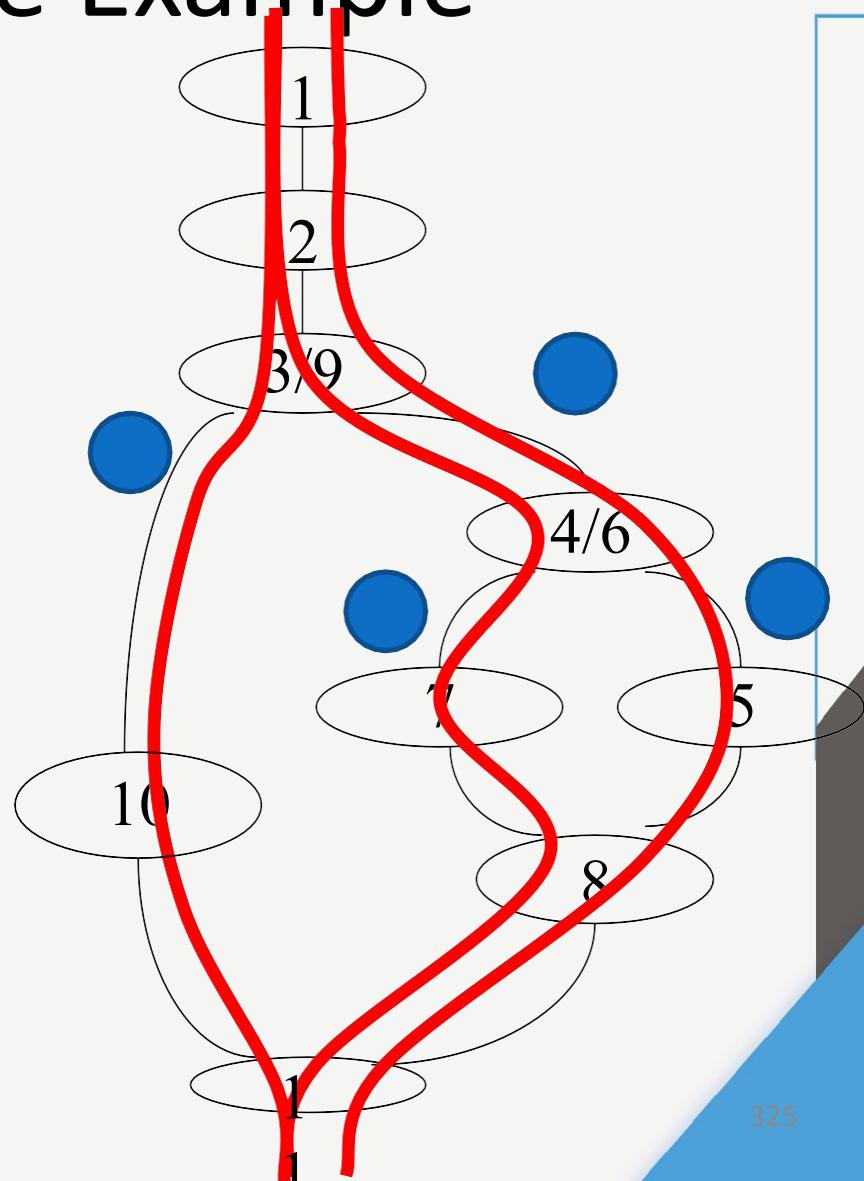
Statement Coverage Example

1. Read bread
2. Read filling
3. If bread = 'Roll' Then
4. If filling = 'Tuna' Then
5. Price = 1.50
6. Else
7. Price = 1.00
8. End If
9. Else
10. Price = 0.75
11. End If



Decision Coverage Example

1. Read bread
2. Read filling
3. If bread = 'Roll' Then
4. If filling = 'Tuna' Then
5. Price = 1.50
6. Else
7. Price = 1.00
8. End If
9. Else
10. Price = 0.75
11. End If



Coverage Answer of Above Ex.

Statement Coverage to achieve 100%

- **Answer is : 3**
- **Reason : Consider 3 inputs for covering 3 statements**
 - Bread = Roll and filling = Tuna, which cover price = 1.50
 - Bread = Roll and filling = Other, which cover price = 1.00
 - Bread = Swiss and filling = Other, which cover price = 0.75

Decision/Branch Coverage to achieve 100%

- **Answer = 3**
- **Reason : Consider 3 inputs for covering 4 branches.**
 - Bread = Roll and filling = Tuna, which cover branch 2 and 4
 - Bread = Roll and filling = Other, which cover branch 2 and 3
 - Bread = Swiss and filling = Other, which cover branch 1.

No of Branches : 4 (T and F of each IF...ELSE)

No of Executable Statements : 7

Other White Box Techniques

Branch Condition testing

- Branch Condition Testing requires that the True and False of each Boolean operand is tested (**Boolean Operands** in this example: *If A > 30 and B >= 5*)

Branch Condition Combination testing

- Branch Condition Combination Coverage would require all **combinations** of Boolean operands to be evaluated

Modified Condition Decision testing

- Modified Condition Decision Coverage requires test cases to show that each Boolean operand can independently affect the outcome of the decision

Dataflow testing

- Data flow testing aims to execute sub-paths from points where each variable in a component is defined to points where it is referenced.

Linear Code Sequence And Jump (LCSAJ) testing

- LCSAJ testing requires a model of the source code which identifies control flow jumps (where control flow does not pass to a sequential statement).

Experience Based Testing

Introduction

- Experience based techniques are non structured and do not rely on specification documents. This makes them unable to be measured in terms of coverage

- In **experience-based techniques**, people's knowledge, skills and background are of prime importance to the test conditions and test cases.

- The experience of both technical and business people is required, as they bring different perspectives to the **test analysis and design process**.

- This may be the only type of technique used for **low-risk systems**, but this approach may be particularly useful under extreme time pressure – in fact this is **one of the factors leading to exploratory testing**.

Introduction(Cont...)

- Uses the knowledge of people and the experience of past projects to determine likely errors based on the knowledge

- Testers
- Users
- Stakeholders

- Good to identify tests which may not be explicitly described in the specification

- Most beneficial when applied after more formal measures

Grey Box Testing

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application.

In software testing, **the term the more you know the better** carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge.

Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database.

Having this knowledge, **the tester is able to better prepare test data and test scenarios when making the test plan.**

Adhoc Testing(Error Guessing)

- Adhoc testing is an informal testing type with an **aim to break the system.**
- It does not follow any test design techniques to create test cases.
- **In fact it does not create test cases altogether!**
- This testing is primarily performed **if the knowledge of testers in the system under test is very high.**
- Testers randomly test the application without any test cases or any business requirement document.
- Adhoc Testing does not follow any structured way of testing and it is randomly done on any part of application.
- **Main aim of this testing is to find defects by random checking.**
- **Adhoc testing can be achieved with the testing technique called Error Guessing.**
- Error guessing can be done by the people having enough experience on the system to “**guess**” the most likely source of errors.

Adhoc Testing(Error Guessing)

The Error guessing is a technique where the experienced and good testers are encouraged to think of situations in which the software may not be able to cope.

Some people seem to be naturally good at testing and others are good testers because they have a **lot of experience** either as a **tester or working with a particular system** and so are able to find out its weaknesses.

This is why an error guessing approach, used after **more formal techniques** have been applied to some extent, can be very effective.

It also saves a lot of time because of the assumptions and guessing made by the experienced testers to find out the defects which otherwise won't be able to find.

Using experience to postulate errors.

Use Error Guessing to Complement Test Design Techniques.

Types of Adhoc Testing

There are different types of Adhoc testing and they are listed as below:

1. Buddy Testing

- Two buddies mutually work on identifying defects in the same module. Mostly one buddy will be from development team and another person will be from testing team. Buddy testing helps the testers develop better test cases and development team can also make design changes early. This testing usually happens after unit testing completion.

2. Pair testing

- Two testers are assigned modules, share ideas and work on the same machines to find defects. One person can execute the tests and another person can take notes on the findings. Roles of the persons can be a tester and scribe during testing.

3. Monkey Testing

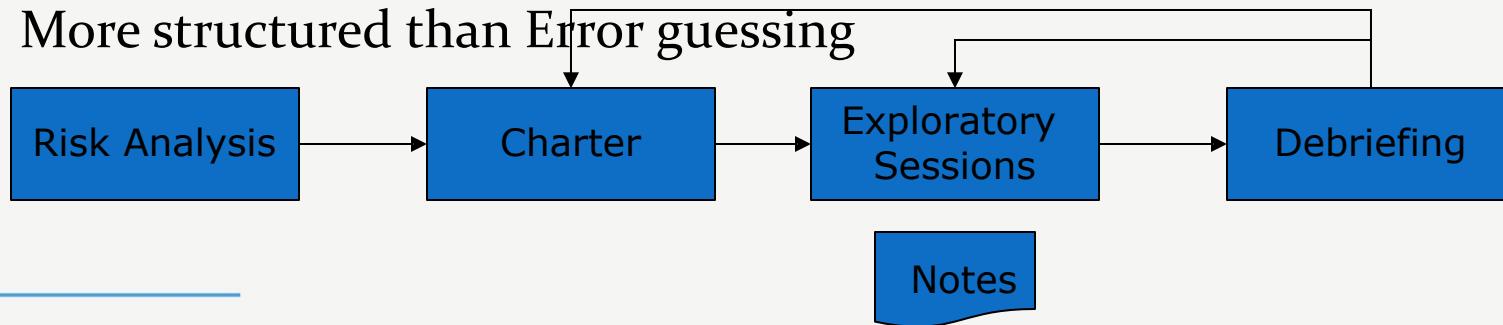
- Randomly test the product or application without test cases **with a goal to break the system**.

Exploratory Testing

Exploratory testing is a concurrent process where

- Test design, execution and logging happen simultaneously
- Testing is often not recorded
- Makes use of experience, heuristics and test patterns
- Testing is based on a test charter that may include
 - Scope of the testing (in and out)
 - The focus of exploratory testing is more on testing as a “thinking” activity.
 - A brief description of how tests will be performed
 - Expected problems
- Is carried out in time boxed intervals

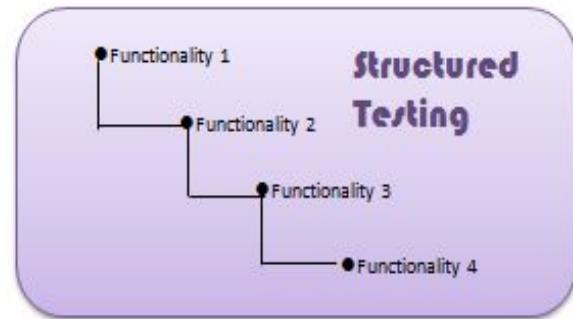
More structured than Error guessing



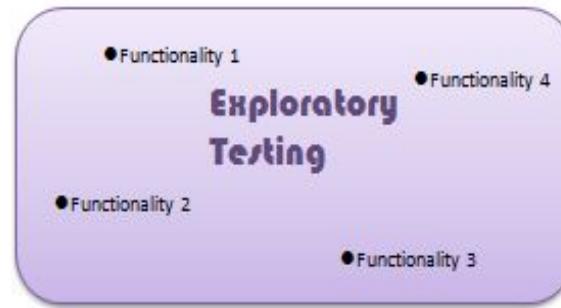
Exploratory Testing

Though the current trend in testing is to push for automation, exploratory testing is a new way of thinking. **Automation has its limits**

- Is not random testing but it is Adhoc testing with purpose of find bugs
- Is structured and rigorous
- Is cognitively (thinking) structured as compared to procedural structure of scripted testing. This structure comes from Charter, time boxing etc.
- Is highly teachable and manageable
- Is not a technique but it is an approach. What actions you perform next is governed by what you are doing currently



**Functionalities are checked
in a structured manner**



**Functionalities are checked
in a ad-hoc manner**

Exploratory vs Scripted Testing

Scripted Testing

Directed from requirements

Determination of test cases well in advance

Confirmation of testing with the requirements

Emphasizes prediction and decision on making

Involves confirmed testing

Is about Controlling tests

Like making a speech – you read from a draft

The script is in control

Exploratory Testing

Directed from requirements and exploring during testing

Investigation of system or application

Emphasizes on adaptability and learning

Involves Investigation

Is about Improvement of test design

Like making a conversion – its spontaneous

The tester's mind is in control

Black Box
(Specification Based)

- Based on requirements
- From the requirements, tests are created
- Specification Models can be used for systematic test case design

Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Tables
- State Transition Testing
- Use Case Testing

White
(Structure Based)

- Based on code and the design of the system
- The tests provide the ability to derive the extent of coverage of the whole application

Techniques

- Statement coverage
- Branch Coverage
- Decision Coverage

Experience Based

- Based on the knowledge of the tester
- Using past experienced use & intuition to “guess” where errors may occur

Techniques

- Grey Box
- Error Guessing
- Exploratory Testing

Smoke and Sanity Testing

Introductions

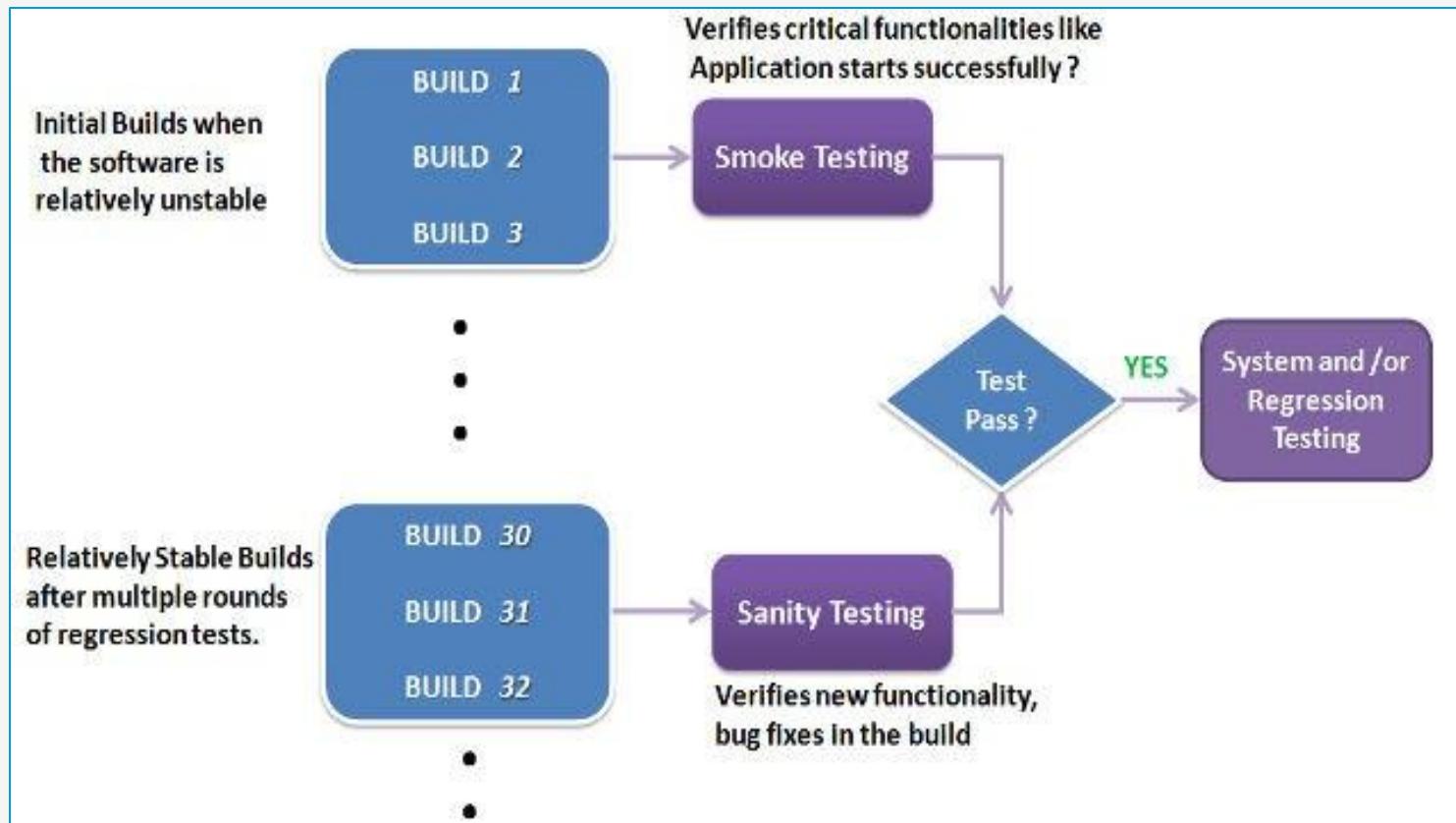
Smoke and Sanity testing are the most misunderstood topics in Software Testing. There is enormous amount of literature on the subject, but most of them are confusing. The following article makes an attempt to address the confusion.

Software Build

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

Introductions

The key differences between Smoke and Sanity Testing can be learned with the help of following diagram –



Smoke Testing

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

Smoke Testing Examples

- **Web Based Testing :**

- New registration button is added in the login window and build is deployed with the new code. We perform smoke testing on a new build.

- **Desktop Based Testing :**

- If applicable in your SUT (System Under Test), as part of the smoke test you should try to successfully login with old and newly created credentials. Also, verify that you are able to successfully log out of the system without any errors

- **Mobile Based Testing :**

- If mobile is in 4g but now new mobile are you buying and new mobile is 5g so we check mobile 4g is working properly or not.

- **Game Based Testing :**

- When we play the game and make it score will be added in score but when we update the application but score not be changed

Sanity Testing

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

Sanity Testing Examples

- **Web Based Testing :**

- In an e-commerce project, main modules are login page, home page, user profile page, user registration etc. There is a defect in the login page when the password field accepts less than four alpha numeric characters and the requirement mentions that this password field should not be below eight characters.

- **Desktop Based Testing :**

- If applicable in your SUT (System Under Test), as part of the smoke test you should try to successfully login with old and newly created credentials. Also, verify that you are able to successfully log out of the system without any errors

- **Mobile Based Testing :**

- Verify the device in different available networks like 2G, 3G, 4G or WIFI.
- Interrupt testing- Able to receive the calls while running the application

- **Game Based Testing :**

- When we play the game so new functionality not affect the other mobile functionality d

Smoke Testing vs Sanity Testing

Smoke Testing Sanity Testing

Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine

The objective of this testing is to verify "stability" of the system in order to proceed with more rigorous testing

This testing is performed by the developers or testers

Smoke testing is usually documented or scripted

Smoke testing is a subset of Regression testing

Smoke testing exercises the entire system from end to end

Smoke testing is like General Health Check Up

Sanity Testing is done to check the new functionality / bugs have been fixed

The objective of the testing is to verify the "rationality" of the system in order to proceed with more rigorous testing

Sanity testing is usually performed by testers

Sanity testing is usually not documented and

is unscripted

Sanity testing is a subset of Acceptance testing

Sanity testing exercises only the particular component of the entire system

Sanity Testing is like specialized health check up

Re-Testing and Regression Testing

Introduction

- The purpose of regression testing is to confirm that a recent program or code change has not adversely affected existing features.

- Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

Confirmation Testing (Re-Testing)

- **Re-testing: Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions**
- Whenever a fault is detected and fixed then the software should be re-tested to show that the original fault has been fixed. This is known as Re-Testing.
- It is important that the test case is repeatable.
- In order to support this the test identifier should be included on the fault report.
- It is important that the environment and test data used are as close as possible as those used during the original test.

Regression Testing

Regression Testing: Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

- If the test is re-run and passes you cannot necessarily say the fault has been resolved because ..
- You also need to ensure that the modifications have not caused unintended side-effects elsewhere and that the modified system still meets its requirements – Regression Testing

Regression Testing

Regression testing should be carried out:

- when the system is stable and the system or the environment changes
- when testing bug-fix releases as part of the maintenance phase
- It should be applied at all Test Levels
- It should be considered complete when agreed completion criteria for regression testing have been met
- Regression test suites evolve over time and given that they are run frequently are ideal candidates for automation

Need of Regression Testing

- Change in requirements and code is modified according to the requirement
 - New feature is added to the software
 - Defect fixing
 - Performance issue fix
-
- ## Difference between Re-Testing and Regression Testing

- Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed, defect needs to be re-opened. If fixed, defect is closed.
- Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

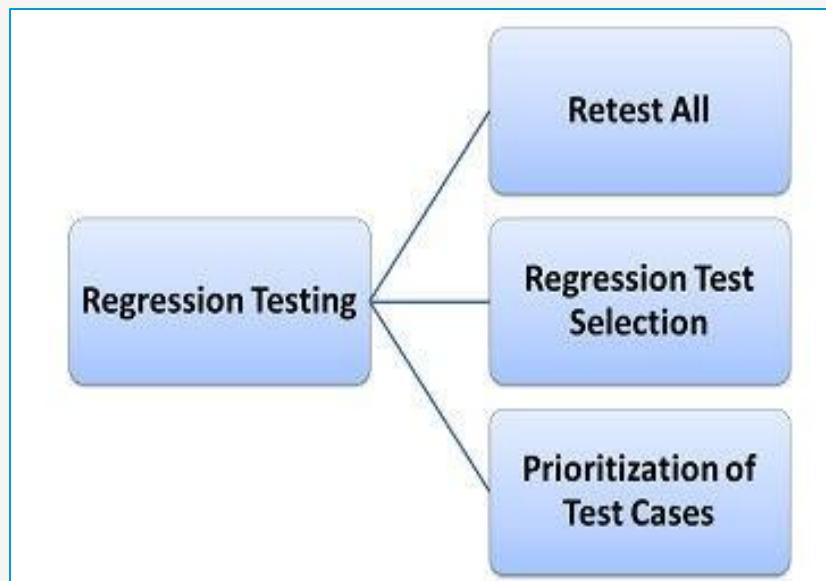
Regression Testing Techniques

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features.

These modifications may cause the system to work incorrectly.

Therefore, Regression Testing becomes necessary.

Regression Testing can be carried out using following techniques:



Regression Testing Techniques

Retest All

- This is one of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of test suite to be run
- Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Prioritization Of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Challenges Regression Testing

- With successive regression runs, test suites become fairly large. Due to time and budget constraints, the entire regression test suite cannot be executed
- Minimizing test suite while achieving maximum test coverage remains a challenge
- Determination of frequency of Regression Tests, i.e., after every modification or every build update or after a bunch of bug fixes, is a challenge.



Regression Testing Tools

- **Quick Test Professional (QTP)**
- **Rational Functional Tester (RFT)**
- **Selenium**

When use the testing tools?

- If your software undergoes frequent changes, regression testing costs will escalate.
- In such cases, Manual execution of test cases increases test execution time as well as costs.
- Automation of regression test cases is the smart choice in such cases.
- Extent of automation depends on the number of test cases that remain reusable for successive regression cycles.

End to End Testing

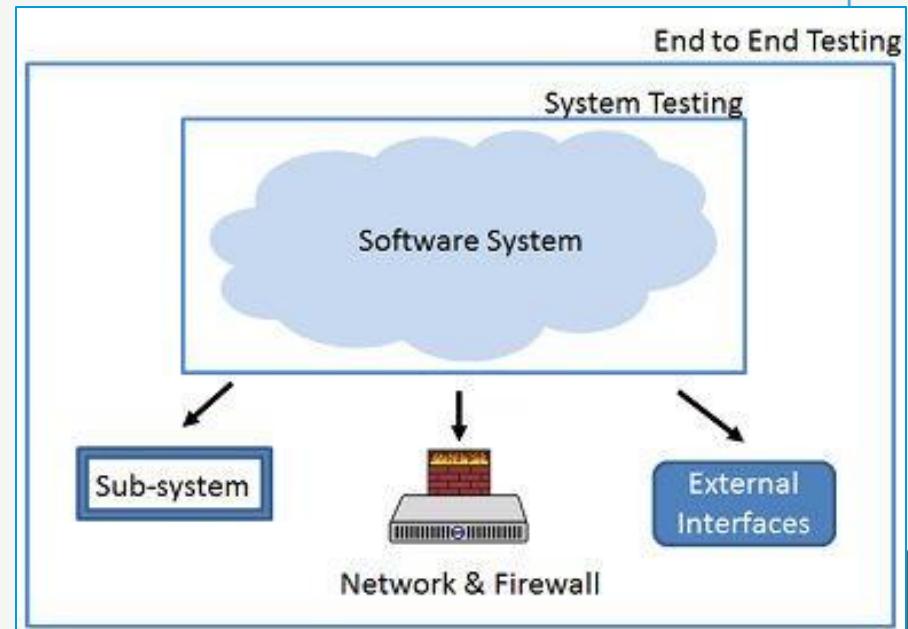
Introduction

- Unlike System Testing, End-to-End Testing not only validates the software system under test but also checks its integration with external interfaces.

- Hence, the name "**End-to-End**". The purpose of End-to-End Testing is to exercise a complete production-like scenario.

- Along with the software system, it also validates batch/data processing from other upstream/downstream systems.

- End to End Testing is usually executed after functional and system testing. It uses actual production like data and test environment to simulate real-time settings. End-to-End testing is also called **Chain Testing**

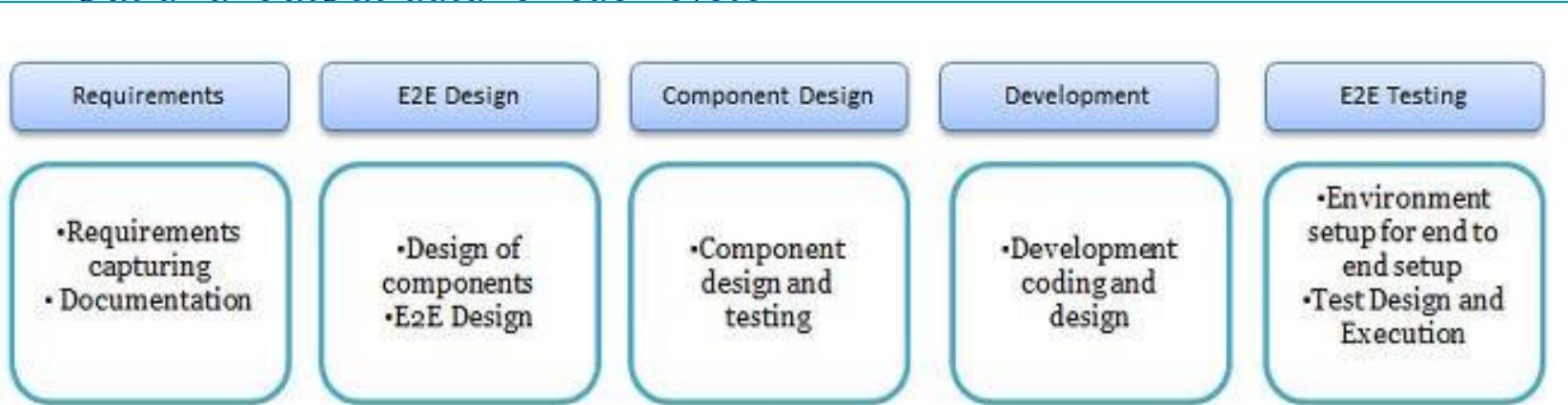


Why End-to End Testing?

- Modern software systems are complex and are interconnected with multiple sub-systems
- A sub-system may be different from the current system or may be owned by another organization.
- **If any one of the sub-system fails, the whole software system could collapse.**
- This is major risk and can be avoided by End-to-End testing.
- End-to-End testing verifies the complete system flow. It increase test coverage of various sub-systems.
- It helps detect issues with sub-systems and increases confidence in the overall software product.

End-to End Testing Process

- Study of end to end testing requirements
- Test Environment setup and hardware/software requirements
- Describe all the systems and its subsystems processes.
- Description of roles and responsibilities for all the systems
- Testing methodology and standards
- End to end requirements tracking and designing of test cases
- Input and output data for each system



End-to End Testing

End to End Testing Design

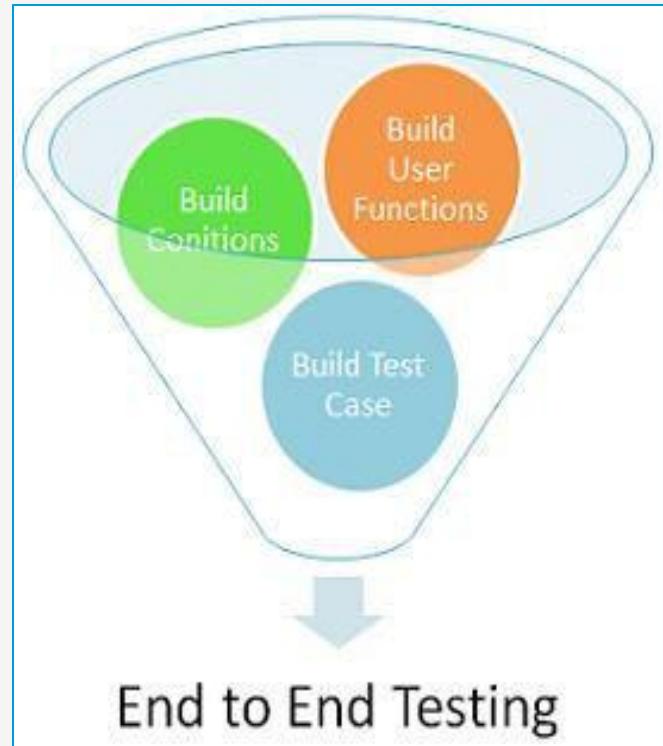
framework

consists of three parts

- Build user functions
- Build Conditions
- Build Test Cases

Metrics used for End to End Testing.

- Test Case preparation status
- Weekly Test Progress
- Defects Status & Details
- Environment Availability



Build User Functions

Following activities should be done as a part of build user functions:

- List down the features of the system and their interconnected components
- List the input data, action and the output data for each feature or function
- Identify the relationships between the functions
- Determine whether the function can be reusable or independent
- For example –Consider a scenario where you login into your bank account and transfer some money to another account from some other bank (3rdparty sub-system)
 - Login into the banking system
 - Check for the balance amount in the account
 - Transfer some amount from your account to some other bank account (3rdparty sub-system)
 - Check the your latest account balance
 - Logout of the application

Build Conditions

Following activities are performed as a part of build conditions:

- Building a set of conditions for each user function defined
- Conditions include sequence, timing and data conditions
- For example –Checking of more conditions like
 - **LOGIN PAGE**
 - Invalid User Name and Password
 - Checking with valid user name and password
 - Password strength checking
 - Checking of error messages
 - **BALANCE AMOUNT**
 - Check the current balance after 24 hours.(If the transfer is sent to different bank)
 - Check for the error message if the transfer amount is greater than the current balance amount
 - **BUILD TEST SCENARIO**
 - Login into the system
 - Check of bank balance amount
 - Transfer the bank balance amount

Build Test Case

- Build one or more test cases for each scenario defined .Test cases may include each condition as single test case.

End to End Testing Examples

- **Web Based Testing :**
 - Type URL into the address bar to launch the Gmail login page.
 - Log into account with valid credentials.
- **Desktop Based Testing :**
 - When punch machine connect with your desktop PC at that time your intime punch will be show in your system
- **Mobile Based Testing :**
 - A proper end-to-end mobile test plan ensures that the tested applications will function as planned on various devices with different screen sizes, resolutions, operating systems and internal hardware, as well as on different carrier networks.
- **Game Based Testing :**
 - :when remote will be connect with your game machine/ play station
 - When AR with connect with any Image or VR box connect with mobile Device

End-to End vs System Testing

End to End Testing

Validates the software system as well as interconnected sub-systems as per the requirements specifications.

It checks the complete end-to-end process and flow. features.

All interfaces, backend systems will be Functional and Non-Functional Testing

considered for testing will be considered for testing

It's executed once system testing is completed. It's executed after integration testing.

End to End testing involves checking external interfaces which can be complex to Both Manual and Automation can be automate. Hence Manual Testing is performed for system testing preferred.

System Testing

Validates just the software system as per the requirements specifications.

It checks system functionalities

Functional and Non-Functional Testing

will be considered for testing

after integration testing.

Manual Testing is performed for system testing preferred.

Positive and Negative Testing

Introduction

- **Software testing** is process of verifying and validating the software or application and checks whether it is working as expected.

- The intent is to find defects and improve the product quality.

- There are two ways to test the software, via, Positive Testing and Negative Testing.

- In both the testing, following needs to be considered:

- Input data
- Action which needs to be performed
- Output Result

TESTING TECHNIQUE USED FOR POSITIVE AND NEGATIVE TESTING:

- Boundary Value Analysis
- Equivalence Partitioning

Positive Testing(Scenario)

- Positive Testing can be performed on the system by providing the valid data input.
- It checks whether an application behaves as expected with the positive input.
- This is to test to check the application that does what it is supposed to do so
- There is a text box in an application which can accept only numbers. Entering values upto 99999 will be acceptable by the system and any other values apart from this should not be acceptable.
- To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values.

Enter Only Numbers

99999

Positive Testing

Negative Testing(Scenario)

- **Negative Testing** can be performed on the system by providing **invalid data as input**. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do so.

- For the example above Negative testing can be performed by testing by entering alphabets characters from A to Z or from a to z.

- Either system text box should not accept the values or else it should throw an error message for these invalid data inputs.

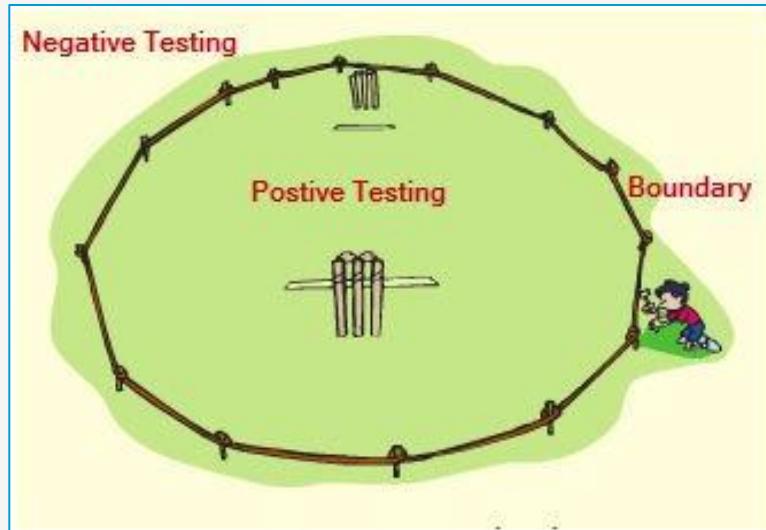
Enter Only Numbers

abcdef

Negative Testing

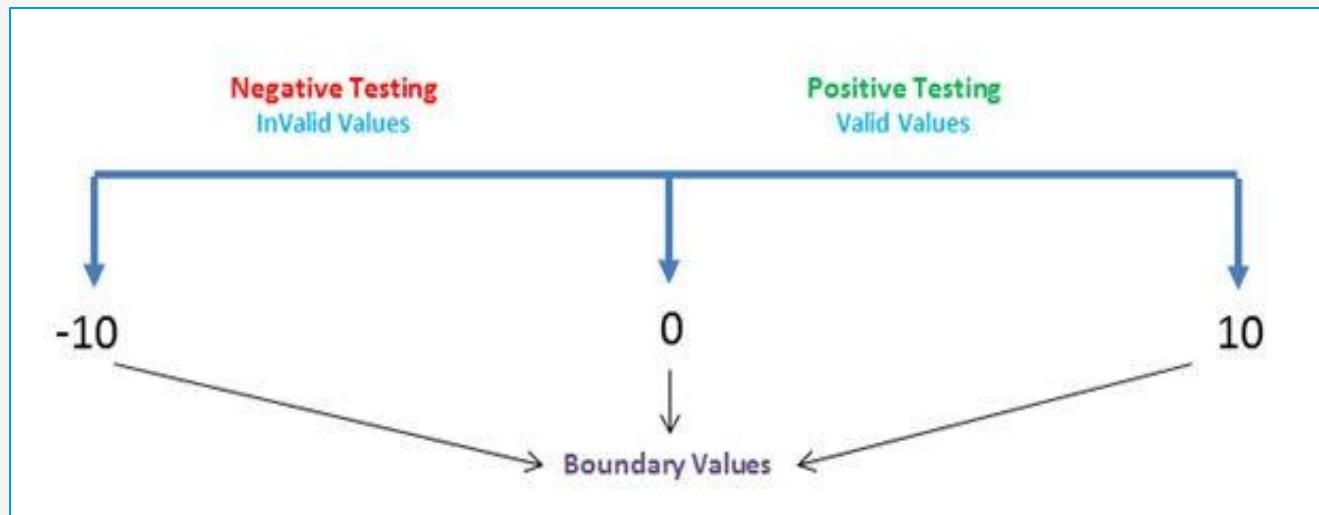
Boundary Value Analysis (BVA)

- This is one of the software testing technique in which the test cases are designed to include values at the boundary.
- If the input data is used within the boundary value limits, then it is said to be Positive Testing.
- If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.



BVA Example

A system can accept the numbers from 0 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0 , 10 and -10 will be tested.

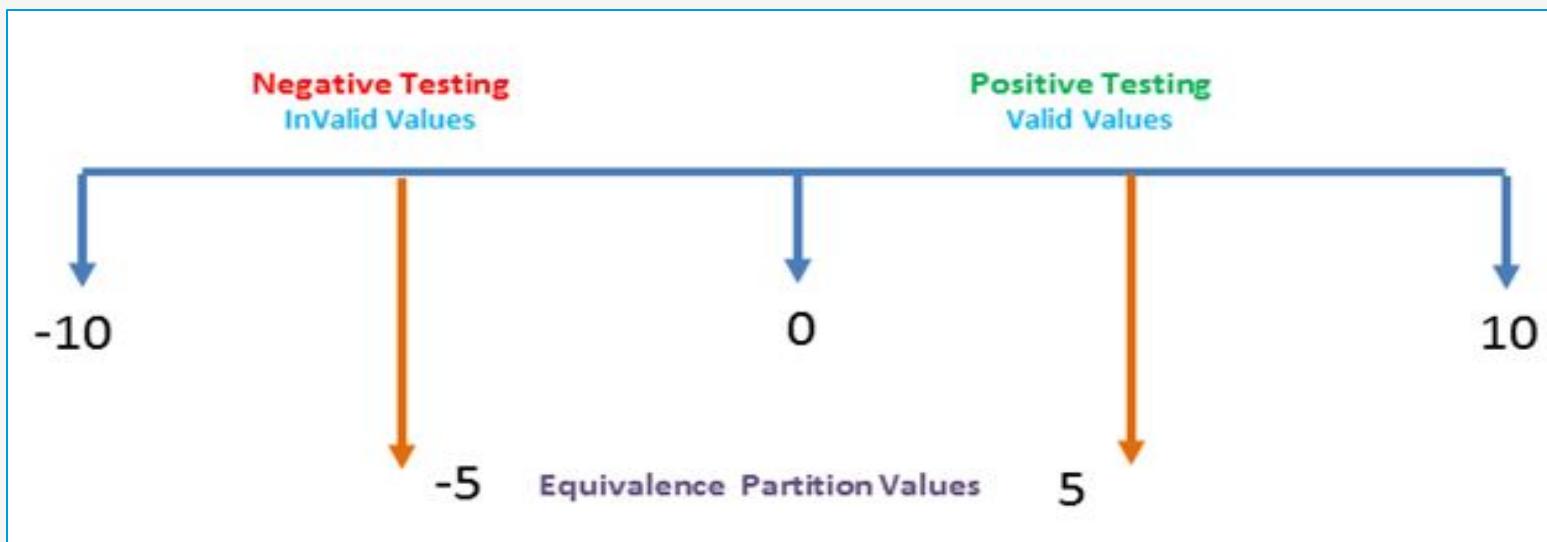


Equivalence Partitioning (EP)

- This is a software testing technique which divides the input date into many partitions .
- Values from each partition must be tested at least once. Partitions with valid values are used for Positive Testing.
- While, partitions with invalid values are used for negative testing.

EP Example

Numeric values Zero to ten can be divided to two (or three) partition. In our case we have two partitions -10 to -1 and 0 to 10. Sample values (5 and -5) can be taken from each part to test the scenarios.



Maintenan ce Testing

Maintenance Testing

Maintenance testing: Testing the changes to an operational system or the impact of a changed environment to an operational system

- testing changes to a Live System

- Triggered by, for example,

- Modification**

- software upgrades
 - Operating system changes
 - system tuning
 - emergency fixes

- Software Retirement** (may necessitate data archiving tests)

- Migration**

- System migration (including operational tests of new environment plus changed software)
 - database migration

Objectives of Maintenance

Testing

- Develop tests to detect problems prior to placing the change into production
- Correct problems identified in the live environment
- Test the completeness of needed training material
- Involve users in the testing of software changes

Concepts of Maintenance

Testing

- will the testing process be planned?

- will testing results be recorded?

- will new faults be introduced into the system?

- will system problems be detected during testing?

- how much regression testing is feasible?

- will training be considered?

Problems of Maintenance

Testing

- All that is available is the source code (usually with poor internal documentation and no record of testing) – poor or missing specifications

- Program structure, global data structures, system interfaces and performance and/or design constraints are difficult to determine and frequently misinterpreted

- Base-lined test plans and/or regression test packs often not updated



How can we test changes?

- Maintenance testing involves testing what has been changed (i.e. **Re-Testing**)
- It also, **importantly**, utilises Impact Analysis as a method for determining what **Regression testing** is required for the whole system
- Traceability of Test-ware to source documents essential for effective impact analysis (we cover this more in a later topic)
- Scope of Maintenance tests based on Risk assessment – including size of change and size of system
- Maintenance testing may involve one or more test levels and one or more test types

How to Choose that which testing technique is best?

Each individual technique is aimed at particular types of defect. For example, state transition testing is unlikely to find boundary defects.

Internal Factor to select Testing Techniques

- Models used in developing the system
- Tester's **knowledge, skill and their experience**
- Similar **type of defects**
- **Test objective**
- **Documentation available**
- Life cycle model used (**Process of Development**)

External Factor to select Testing Techniques

- **Customer and contractual requirements**
- Risk assessment (**Level of Risk and Types of Risk**)
- Type of system used
- Regulatory requirements
- **Time and budget of the project**



Static Testing & Technique

Static Testing

Static Testing: Static testing techniques involve examination of the project's documentation, software and other information about the software products without executing them

- Static Testing Includes both Reviews (e.g. of documentation) and Static Analysis of code
- Reviews, Static Analysis and dynamic testing have the same objective – identifying defects
- Static Testing and Dynamic Testing are complementary each technique can find different types of defects effectively and efficiently

Dynamic testing: Testing that involves the execution of the software of a component or system.



Static Testing(Cont...)

Static testing techniques rely on the manual examination (reviews) and automated analysis (static analysis) of the code or other project documentation.

Reviews are a way of testing software work products (including code) and can be performed well before dynamic test execution.

Defects detected during reviews early in the life cycle are often much cheaper to remove than those detected while running tests (e.g. defects found in requirements).

A review could be done entirely as a manual activity, but there is also tool support available.

The main manual activity is to examine a work product and make comments about it.

Static Testing(Cont...)

- Any software work product can be reviewed, including requirements specifications, design specifications, code, test plans, test specifications, test cases, test scripts, user guides or web pages.

- Benefits of reviews include early defect detection and correction, development productivity improvements, reduced development timescales, reduced testing cost and time, lifetime cost reductions, fewer defects and improved communication.

- Compared to dynamic testing, static techniques find causes of failures (defects) rather than the failures themselves.

- Typical defects that are easier to find in reviews than in dynamic testing are: deviations from standards, requirement defects, design defects, insufficient maintainability and incorrect interface specifications.

Use of Static Testing

Since static testing can start early in the life cycle so early feedback on quality issues can be established.

As the defects are getting detected at an early stage so the rework cost most often relatively low.

Development productivity is likely to increase because of the less rework effort.

Types of the defects that are easier to find during the static testing are:

- deviation from standards,
- missing requirements,
- design defects,
- non-maintainable code,
- inconsistent interface specifications.

Informal Reviews

- Informal reviews are applied many times during the early stages of the life cycle of the document.
- A two person team can conduct an informal review. In later stages these reviews often involve more people and a meeting.
- The goal is to keep the author and to improve the quality of the document.
- The most important thing to keep in mind about the informal reviews is that they are **not documented**.

Formal Reviews

- Formal reviews follow a formal process. It is well structured and regulated.

- A formal review process consists of six main steps:

- Planning
- Kick-off
- Preparation
- Review meeting
- Rework
- Follow-up

Formal Review - Planning

- The first phase of the formal review is the Planning phase.
- In this phase the review process begins with a request for review by the author to the moderator (or inspection leader).
- A moderator has to take care of the scheduling like date, time, place and invitation of the review.
- For the formal reviews the moderator performs the entry check and also defines the formal exit criteria.
- The **entry check** is done to ensure that the reviewer's time is not wasted on a document that is not ready for review.
- After doing the entry check if the document is found to have very little defects
then it's ready to go for the reviews.

Formal Review - Planning

So, the **entry criteria** are to check that whether the document is ready to enter the formal review process or not. Hence the entry criteria for any document to go for the reviews are:

- The documents should not reveal a large number of major defects.
- The documents to be reviewed should be with line numbers.
- The documents should be cleaned up by running any automated checks that apply.
- The author should feel confident about the quality of the document so that he can join the review team with that document.

Once, the document clear the entry check the moderator and author decides that which part of the document is to be reviewed.

Since the human mind can understand only a limited set of pages at one time so in a review the maximum size is between 10 and 20 pages.

Hence checking the documents improves the moderator ability to lead the meeting because it ensures the better understanding.

Formal Review – Kick Off

- This kick-off meeting is an optional step in a review procedure.
- The goal of this step is to give a short introduction on the objectives of the review and the documents to everyone in the meeting.
- The relationships between the document under review and the other documents are also explained, especially if the numbers of related documents are high.
- At customer sites, we have measured results up to 70% more major defects found per page as a result of performing a kick-off.

Formal Review – Preparation

- In this step the reviewers review the document individually using the related documents, procedures, rules and checklists provided.
- Each participant while reviewing individually identifies the defects, questions and comments according to their understanding of the document and role.
- After that all issues are recorded using a logging form.
- The success factor for a thorough preparation is the number of pages checked per hour. This is called the **checking rate**.
- Usually the checking rate is in the range of 5 to 10 pages per hour.

Formal Review – Review Meeting

The review meeting consists of **three phases**:

- **Logging phase:**
 - In this phase the issues and the defects that have been identified during the preparation step are logged page by page.
 - The logging is basically done by the author or by a **scribe**.
 - Scribe is a separate person to do the logging and is especially useful for the formal review types such as an inspection.
 - Every defects and its severity should be logged in any of the three severity classes given below:
 - **Critical:** The defects **will cause** downstream damage.
 - **Major:** The defects **could cause** a downstream damage.
 - **Minor:** The defects are **highly unlikely to cause** the downstream damage.
 - During the logging phase the moderator focuses on logging as many defects as possible within a certain time frame and tries to keep a good logging rate (number of defects logged per minute).
 - In formal review meeting the good logging rate should be between one and two defects logged per minute.

Formal Review – Review Meeting

The review meeting consists of **three phases**:

Discussion phase:

- If any issue needs discussion then the item is logged and then handled in the discussion phase.
- As chairman of the discussion meeting, the moderator takes care of the people issues and prevents discussion from getting too personal and calls for a break to cool down the heated discussion.
- The outcome of the discussions is documented for the future reference.

Decision phase:

- At the end of the meeting a decision on the document under review has to be made by the participants, sometimes based on formal **exit criteria**.
- **Exit criteria** are the average number of critical and/or major defects found per page (for example no more than three critical/major defects per page).
- If the number of defects found per page is more than a certain level then the document must be reviewed again, after it has been reworked.

Formal Review – Follow Up

- In this step the moderator check to make sure that the author has taken action on all known defects.
- If it is decided that all participants will check the updated documents then the moderator takes care of the distribution and collects the feedback.
- It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

Formal Review –

- In this step if the number of defects found per page exceeds the certain level then the document has to be reworked.
- Not every defect that is found leads to rework.
- It is the author's responsibility to judge whether the defect has to be fixed.
- If nothing can be done about an issue then at least it should be indicated that the author has considered the issue.

Roles & Tasks During Review Process

The moderator:

- Also known as review leader
- Performs entry check
- Follow-up on the rework
- Schedules the meeting
- Coaches other team
- Leads the possible discussion and stores the data that is collected

The author:

- Illuminate the unclear areas and understand the defects found
- Basic goal should be to learn as much as possible with regard to improving the quality of the document.

Roles & Tasks During Review Process

The scribe:

- Scribe is a separate person to do the logging of the defects found during the review.

The reviewers:

- Also known as checkers or inspectors
- Check any material for defects, mostly prior to the meeting
- The manager can also be involved in the review depending on his or her background.

The managers:

- Manager decides on the execution of reviews
- Allocates time in project schedules and determines whether review process objectives have been met

Types of Review - Walkthrough

Walkthrough:

- It is not a formal process
- It is led by the authors
- Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
- Useful for the people if they are not from the software discipline, who are not used to
 - **or cannot easily understand software development process.**
- Is especially useful for higher level documents like requirement specification, etc.

The goals of a walkthrough:

- To present the documents both within and outside the software discipline in order to gather the information regarding the topic under documentation.
- To explain or do the knowledge transfer and evaluate the contents of the document
- To achieve a common understanding and to gather feedback.
- To examine and discuss the validity of the proposed solutions

Types of Review – Technical

Technical Review:

- It is less formal review
- It is led by the trained moderator but can also be led by a technical expert
- It is often performed as a peer review without management participation
- Defects are found by the experts (such as architects, designers, key users) who focus on the content of the document.
- In practice, technical reviews vary from quite informal to very formal

The goals of the Technical Review are:

- To ensure that at an early stage the technical concepts are used correctly
- To assess the value of technical concepts and alternatives in the product
- To have consistency in the use and representation of technical concepts
- To inform participants about the technical content of the document

Types of Review – Inspection

Inspection:

- It is the most formal review type
- It is led by the trained moderators
- During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting
- It involves peers to examine the product
- A separate preparation is carried out during which the product is examined and the defects are found
- The defects found are documented in a logging list or issue log
- A formal follow-up is carried out by the moderator applying exit criteria

The goals of inspection are:

- It helps the author to improve the quality of the document under inspection
- It removes defects efficiently and as early as possible
- It improves product quality
- It creates common understanding by exchanging information
- It learns from defects found and prevent the occurrence of similar defects

Estimation Techniques

Estimating effort for test is one of the major and important tasks in SDLC. Correct estimation helps in testing the Software with maximum coverage. This section describes some of the techniques which can be useful during the estimating of effort for testing.

Functional Point Analysis

- This method is based on the analysis of functional user requirements of the Software with following categories:
 - Outputs
 - Inquiries
 - Inputs
 - Internal files
 - External files

Test Point Analysis

- This estimation process is used for function point analysis for Black box or Acceptance testing.
- It is use the main elements of this method are: Size, Productivity, Strategy, Interfacing, Complexity and Uniformity etc.

Estimation Techniques

Mark-II method

- It is estimation method used for analysis and measuring the estimation based on end user functional view. The procedure for Mark-II method is:
 - Determine the View Point
 - Purpose and Type of Count
 - Define the Boundary of Count
 - Identify the Logical transactions
 - Identify and Categorize Data Entity Types
 - Count the Input Data Element Types
 - Count the Functional Size

Miscellaneous

You can use other popular estimation techniques like:

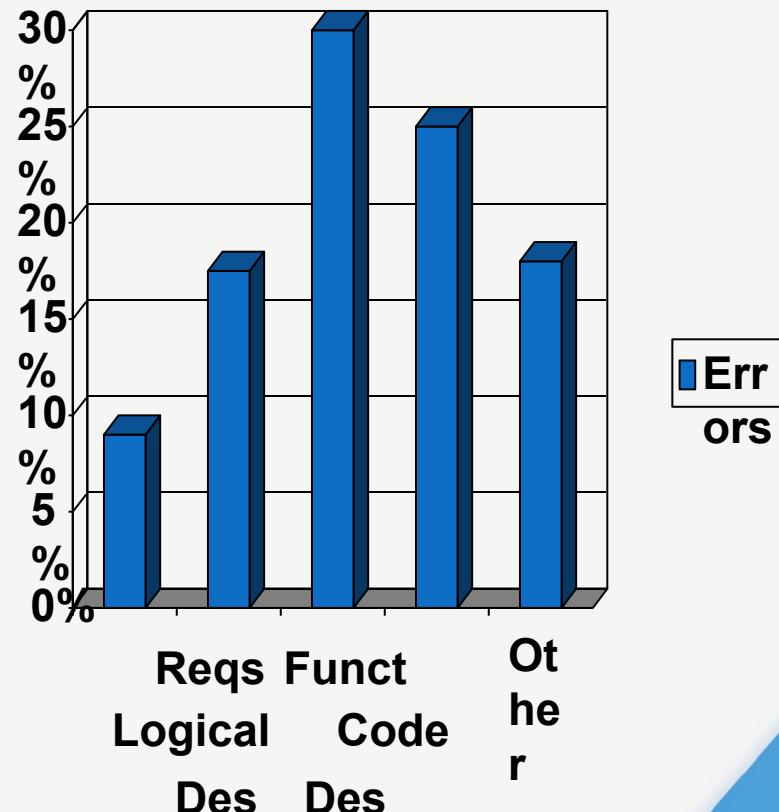
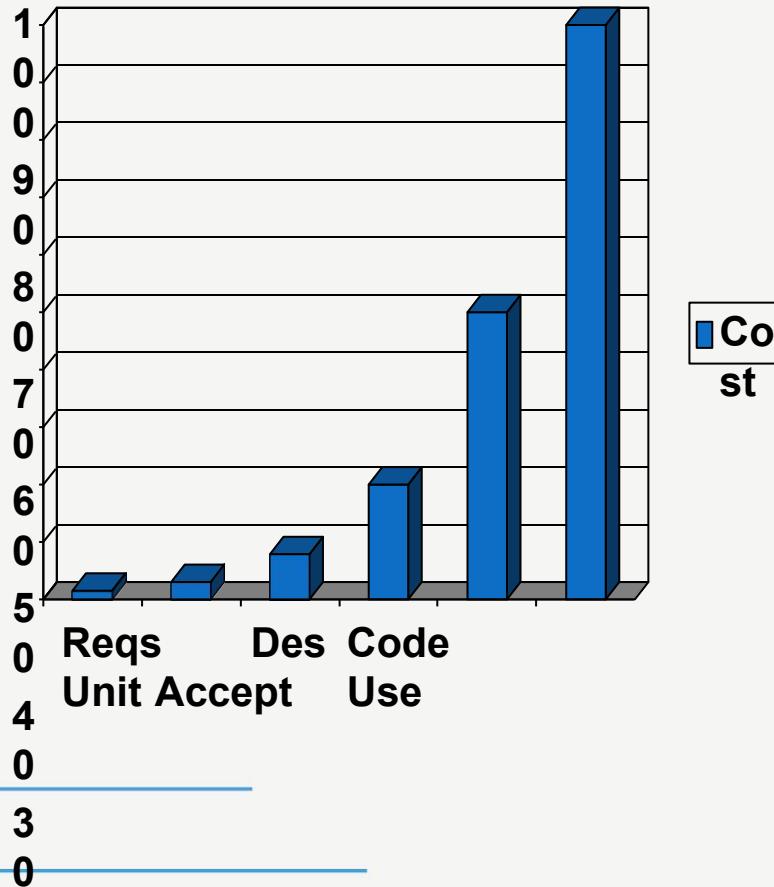
- Delphi Technique
- Analogy Based Estimation
- Test Case Enumeration Based Estimation
- Task (Activity) based Estimation
- IFPUG method

Why Reviews and Test Process?

- Improve the specification, design and code
- Education and training of developers and other project staff
- Determine the root causes of defects and measures for preventing recurrence
- Test and improve standards and procedures
- To assess the early stages of development for progress and completeness
- Reduce the errors in delivered systems
- Learning experience in standards and techniques
- Team building and motivation
- Find errors early in the development lifecycle (more cost effective to fix) More than 60% of the errors in a component can be detected using informal inspections
- To verify and review the coding standards, structure and metrics.

Why Reviews and Test Process?

The Cost of Errors Where Errors Introduce

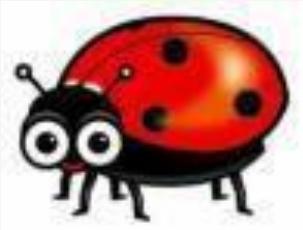


When and What to Reviews?

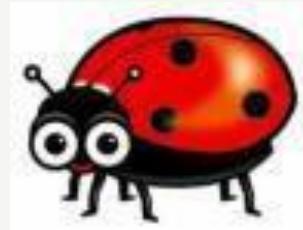
- A review could be done entirely as a manual activity
- Or there is tool support available
- The main manual activity is to examine a work product and make comments about it.
- Any software work product can be reviewed, including:
 - requirements and design specifications
 - source code listings
 - test plans, test cases, test scripts
 - user documentation
 - application administration and support material
 - Web pages
- A software work product can be reviewed one or more times and using one or more review types
- Review everything as soon as possible
- Reviews start well before Dynamic Test execution

What do Review Find?

- In contrast to dynamic testing, reviews find **defects** rather than **failures**



Bugs!



- Typical defects that are easier to find in reviews than in dynamic testing are:

- deviations from standards
- requirement defects
- design defects
- insufficient maintainability
- incorrect interface specifications.

Types of Defect Found By Review

- Unreachable code
- Undeclared variables
- Parameter type mismatches
- Uncalled functions and procedures
- Possible array bound violations
- Security Violations
- Inconsistent interface between modules and components
- Incorrect variable usage
- Syntax checking
- Violations of code standards
- Use of variables without first defining them
- variables that are declared but never used
- Use of variables after they have been “killed”

Testing Deliverable Terms

- Test Set
- Test Case
- Test Condition
- Test Script/Test Step/Test Procedure
- Test Data
- Test Environment

Agile Testing

What is not

AGILE?

Conducting meetings

The team conducts frequent meetings for 10-15 minutes daily, and they think that conducting frequent meetings will be Agile. However, only the following meetings will not be Agile.

Requirements changing anytime

Requirements can be changed at any time, then it is not Agile. For example, a client wants to add some new features and want the changes to be updated at the same time, then this will not be Agile.

Unstructured development

Suppose you are not following any plan and you are working on Adhoc basis then it is not Agile wherein Adhoc testing, testers randomly test the application without following any documents and test design techniques.

No documentation

If the company does not make the documentation, then it is not Agile.

What is Agile?

The **Agile methodology** is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the **work** begins, teams cycle through a process of planning, executing, and evaluating.

- Agile is a philosophy, i.e., a set of values and principles to make a decision for developing software.
- Agile is based on the iterative-incremental model. In an incremental model, we create the system in increments, where each increment is developed and tested individually.

What are

values?

Individuals and interactions, Over processes and tools

Suppose the team finds any issue in software then they search for another process or tool to resolve the issue. But, in Agile, it is preferable to interact with client, manager or team regarding issue and make sure that the issue gets resolved.

Working software, Over comprehensive documentation

Documentation is needed, but working software is much needed. Agile is not saying that documentation is not needed, but working software is much needed. For example, you have 20-page documents, but you do not have a single prototype of the software. In such a case, the client will not be happy because, in the end, the client needs a document.

Customer collaboration, Over contract negotiation

Contract negotiation is important as they make the budget of software, but customer collaboration is more important than over contract negotiation. For example, if you stuck with the requirements or process, then do not go for a contract which we have negotiated. You need to interact with the customer, gather their requirements.

Responding to change, over following a plan

In the waterfall model, everything is planned, i.e., at what time, each phase will be completed. Sometimes you need to implement the new requirements in the middle of the software, so you need to be versatile to make changes in the software.

Agile Principles

- **Customer satisfaction through early and continuous software delivery** – Customers are happier when they receive working software at regular intervals, rather than waiting extended periods of time between releases.
- **Accommodate changing requirements throughout the development process** – The ability to avoid delays when a requirement or feature request changes.
- **Frequent delivery of working software** – Scrum accommodates this principle since the team operates in software sprints or iterations that ensure regular delivery of working software.
- **Collaboration between the business stakeholders and developers throughout the project** – Better decisions are made when the business and technical team are aligned.
- **Support, trust, and motivate the people involved** – Motivated teams are more likely to deliver their best work than unhappy teams.
- **Enable face-to-face interactions** – Communication is more successful when development teams are co-located

- **Working software is the primary measure of progress** – Delivering functional software to the customer is the ultimate factor that measures progress.
- **Agile processes to support a consistent development pace** – Teams establish a repeatable and maintainable speed at which they can deliver working software, and they repeat it with each release.
- **Attention to technical detail and design enhances agility** – The right skills and good design ensures the team can maintain the pace, constantly improve the product, and sustain change.
- **Simplicity** – Develop just enough to get the job done for right now.
- **Self-organizing teams encourage great architectures, requirements, and designs** – Skilled and motivated team members who have decision-making power, take ownership, communicate regularly with other team members, and share ideas that deliver quality products.
- **Regular reflections on how to become more effective** – Self-improvement, process improvement, advancing skills, and techniques help team members work more efficiently.

Scrum

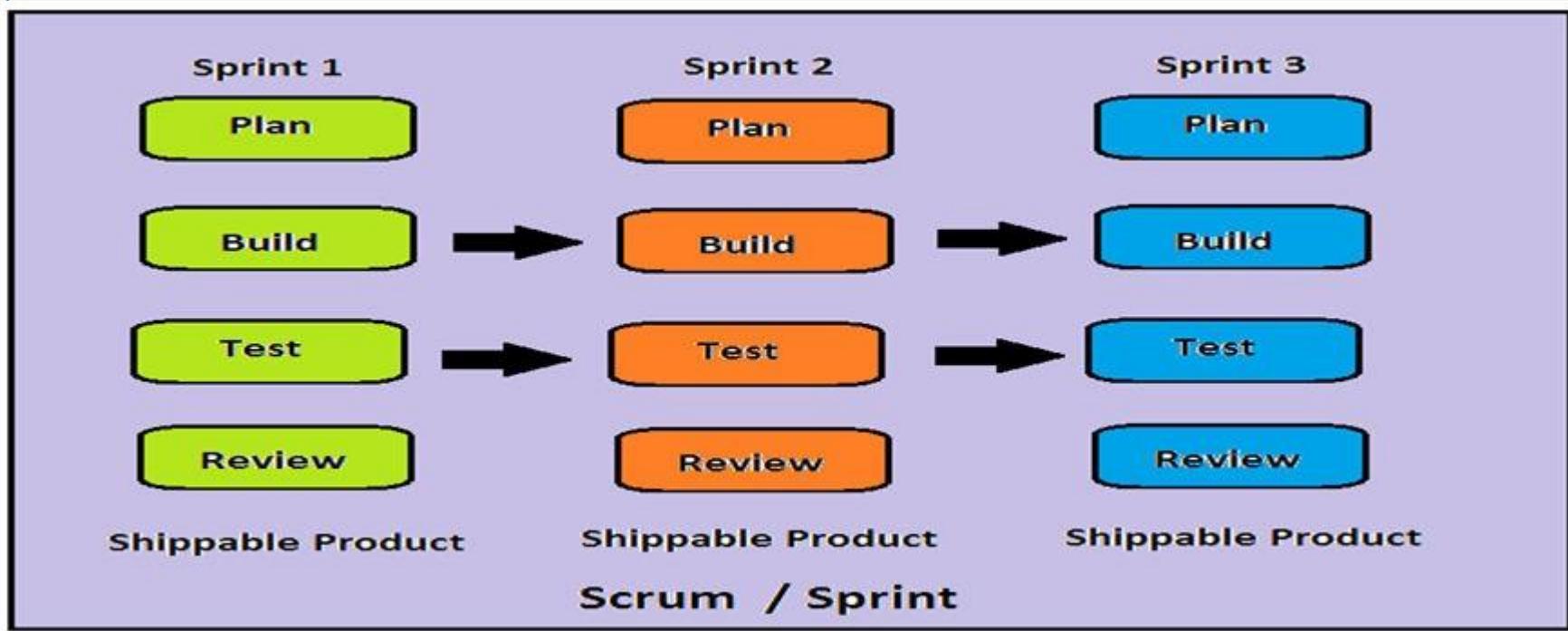
We have studied the Agile methodology where Agile is a set of beliefs which should be followed to develop the software development project. On these beliefs or values, there are many models developed, and in which one of the models is a scrum.

Before going to deep in Scrum, you should know the meanings of some basic terms:

- **Scrum:** Scrum is an agile framework that helps you to organize, iterate, and continue the same project that you are working on. In scrum, a product is built in the series of iterations known as sprints or parts.

- **Sprint:** Sprint is a time-boxed period in which the scrum team needs to finish the set amount of work. Each sprint has a specified timeline, i.e., 2 weeks to 1 month. The scrum team agrees with this timeline during the sprint planning meeting.

- Scrum Master:** Scrum Master is defined as a facilitator or servant-leader to the Scrum development team. Scrum Master must ensure that scrum principles are followed.
- Scrum development team:** A scrum development team is a collection of individual members that includes developers, QA, and scrum master. It decides and provides the effort estimate. The recommended size of the scrum team is between 5 and 9 members.



Artifacts of Scrum

The documentation and stuff which are prepared in scrum are known as Artifacts.

Product Backlog

Product Backlog is a collection of activities that need to be done within the project. When we want to develop software, then we need to perform the 'n' number of activities. For example, we need to develop the e-commerce website then we have to do the 'n' number of activities such we need to create the login page, payment system, cart system, etc. and these 'n' number of activities which are needed to develop the software is known as the product backlog.

Sprint Backlog

We know that in a scrum, we break the scrum into 'n' number of sprints and the objective of a sprint is to bring the small functionality of the software and ship it to the client for demo. In Product backlog, we have to do all the activities which are required to develop the software while in the sprint backlog, a small set of product backlog activities are performed within that sprint. The 'n' number of sprint backlogs is equal to the 1 product backlog.

Burndown Chart

Burndown chart is the outcome of the sprint, which shows the progress in a sprint. After each sprint, we need to examine the progress of each sprint. The burndown chart tells how you are working on the sprint. In the burndown chart, the graph starts from some time, i.e., where the activity gets started, and at the end of the sprint, the graph reaches to zero where the activity ends. It is generally an inclined line from top to bottom.

Scrum Roles

There are three scrum roles:

Product Owner

There is a client who wants to develop his software, so he approaches to the company who can develop his software. What does the company do? The Company assigns a role, i.e., Product Owner. Product Owner is the person who communicates with the clients understands their requirements. Product Owner is the responsible person from the company for software development.

Scrum Master

During the sprint, Agile says that the team should meet together once daily. When the team is following scrum means that they are conducting meetings daily for 10 to 15 minutes. This meeting is known as a scrum meeting. Scrum Master is the person who handles the scrum meeting.

Team

The team comprises of persons who work on the project. It can be developers, testers or designers. When we talk about Agile or Scrum then we talk about the team, we do not talk about developers, or testers as an individual. Agile says that developers can work as a tester or testers can work as a developer when the need arises.

Scrum Ceremonies

Let's look at the following Scrum Ceremonies:

Sprint Planning

Scrum consists of a number of sprints which have a different set of modules used to deliver the software. Before starting the sprint planning, we have a meeting known as sprint planning, and in sprint planning, we discuss what we are going to do in a sprint. In sprint planning, product owner discusses about each feature of a product and estimates the effort involved by the development team.

Daily Scrum

In Scrum, meetings are conducted daily for 15 minutes by Scrum Master, where Scrum Master is the person who manages the meeting. Meeting consists of scrum master, developers, testers, designers, product owner, the client where product owner and client are optional.

Sprint Review

After the completion of each sprint, the meeting is conducted with a client in which a product is shown to the client for demo and team discuss the features they added in the project.

Sprint Backlog

Sprint Backlog is a set of activities that need to be performed within this sprint. Out of the Product Backlog, a set of activities are captured in a sprint backlog, and each activity of a sprint backlog is assigned to a specific person. The minimum time to complete the sprint is 4 days, but it can be stretched to 2-3 weeks.

Sprint

After Sprint Backlog, the team starts working on a sprint, and it can take around 1 to 3 weeks to complete the sprint. The completion of sprint varies from project to project. When sprint gets started, a daily meeting is conducted known as Daily Scrum, and the Scrum Master conducts this meeting. In Daily Scrum, a meeting is conducted daily, and the meeting can be stretched to 10-15 meetings. Meeting has a predefined format, i.e., a team member has to tell what he was doing yesterday, what he will do today, and what are the things which are hampering him to complete his work. It's the responsibility of the scrum master to resolve the issues faced by the team members.

Sprint delivery

When a sprint is completed, then the sprint is delivered to the client. The product is delivered to the client means that minimum set of product backlog known as sprint backlog is completed. The sprint delivery is done so that the client can view the product, it's not that we have developed something and client cannot view.

Sprint Review and Retrospective

Once the sprint delivery is over, there are two types of meetings held, i.e., sprint review and retrospective. The **sprint review** is a meeting in which team members sit together, and they give the demo to the client about the product that they have developed in this sprint.

- A **retrospective** is another meeting which is held between team members. In this meeting, they discuss what is right in this sprint and what went wrong in this sprint, such as the issues hampering their work.
- After sprint review, we go back to the Product Backlog and then sprint planning is done to select the sprint backlog, i.e., sprint2, in this way, this cycle goes on until and unless the whole product is developed and shipped to the client.

Scrum Board

- **Product Backlog:** Product Backlog is a set of activities that need to be done to develop the software.
- **Sprint Backlog:** Sprint Backlog is a backlog that has taken some of the activities from the product backlog which needs to be completed within this sprint.
- **Scrum Board:** Scrum Board is a board that shows the status of all the activities that need to be done within this sprint.

Scrum Board consists of four status:

Open

The 'Open' status means that the tasks which are available in 'Open' are not yet started.

In progress

The 'In progress' status means the developers completed their tasks.

Testing

The 'testing' means that the task is in a testing phase.

Closed

The 'closed' means the task has been completed.

The Agile: Scrum Framework at a glance

Inputs from Executives,
Team, Stakeholders,
Customers, Users



Product Owner



The Team

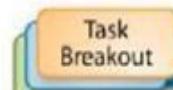


Product Backlog



Sprint Planning Meeting

Estimating Stories
(Story points)



Sprint Backlog



Sprint end date and team deliverable do not change



Sprint Review



Finished Work

Final Demo to PO/QA/DEV

- 1) What went well?
- 2) What went wrong?
- 3) Improvement Areas



JIRA

**(Defect tracking and Management
Tool)**



JIRA

Introduction

- JIRA is a tool developed by Australian Company Atlassian.
- It is used for bug tracking, issue tracking, and project management.
- The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla".
- The basic use of this tool is to track issue and bugs related to your software and Mobile apps.
- The JIRA dashboard consists of many useful functions and features which make handling of issues easy



Log in to your account

Enter email

Continue

OR



Continue with Google



Continue with Microsoft



Continue with Apple



Continue with Slack

Can't log in? • Sign up for an account

 ? CC Settings

Switch to [Show all products](#)

[!\[\]\(b3ea14ecfd4cf9714be031b1041b8053_img.jpg\)](#) [!\[\]\(4dd22c7b72386be6690834f392e1d4f0_img.jpg\)](#) [!\[\]\(48bf315d29d2bee2e1e1eeb963b1ce2a_img.jpg\)](#) [!\[\]\(1ed8529edf7c095b2c88186f660b9d5b_img.jpg\)](#) [!\[\]\(a20c084e9a69af5f7ed6ea9e7adb91bd_img.jpg\)](#) [!\[\]\(350afaa769777dd89fbb87626f7d5504_img.jpg\)](#) [!\[\]\(82cbc8254f4224d4c34b00f201ebcef7_img.jpg\)](#) [!\[\]\(7202ca3667ac94c7c211a5bdb1dfa3d1_img.jpg\)](#) [!\[\]\(82be1f279f1a03fa9229a9832a861c7b_img.jpg\)](#) [!\[\]\(7117b7e11f2acdfc54d114e3e6931424_img.jpg\)](#)

Jira Software
rahulsanghavi

Jira Software
topstech

Confluence
[TRY](#)

Jira Service Management

Opsgenie

Account settings

Atlassian Support

Atlassian Community

Self-managed licensing

Administration

Frequent



DealMart1
Jira



E-mobile shoap
Jira



Banking System
Jira

X

Project templates

Project templates

Software development

Service management

Work management

Marketing

Human resources

Finance

Design

Personal

Operations

Legal

Sales

PRODUCTS

Jira Software

Jira Service Management

Jira Work Management

Software development

Plan, track and release great software. Get up and running quickly with templates that suit the way your team works. Plus, integrations for DevOps teams that want to connect work across their entire toolchain.



Kanban

Visualize and advance your project forward using issues on a powerful board.



Scrum LAST CREATED

Sprint toward your project goals with a board, backlog, and roadmap.



Bug tracking

Manage a list of development tasks and bugs.

[← Back to project types](#)

Add project details

You can change these details anytime in your project settings.

Name *

E-Commerce

Access Anyone with access to rahulsanghavi can access and administer this project. Upgrade your plan to customize project permissions.

Key 1*

COM

Connect repositories, documents, and more

Sync your team's work from other tools with this project for better visibility, access, and automation.

Template

Change template



Scrum

Sprint toward your project goals with a board, backlog, and roadmap.

Type

Change type



Team-managed

Control your own working processes and practices in a self-contained space.

Cancel

Create project

Jira Software Your work Projects Filters Dashboards People Apps Create

Search

E-Commerce Software project

Roadmap

Backlog

Board

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / E-Commerce

COM board

Search CC User

GROUP BY None

TO DO IN PROGRESS DONE



You haven't started a sprint

You can't do anything on your board because you haven't started a sprint yet. Go to the backlog to plan and start a sprint.

Go to Backlog

Quickstart

Jira Software Your work ▾ Projects ▾ Filters ▾ Dashboards ▾ People ▾ Apps ▾ Create Search Help Settings CC

E-Commerce
Software project

Roadmap

Backlog

Board

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / E-Commerce

Backlog

Issues without epic

for a new e-commerce website to launch, the highest business value will be when a new user is able to buy an item from website.

What will the Epic be called?

Backlog (0 issues)

Your backlog is empty.

+ Create issue

0 0 0 Create sprint

Insights

Quickstart

Jira Software Your work Projects Filters Dashboards People Apps Create Q Search ⚙️ 🌐 🔍

E-Commerce Software project Roadmap Backlog Board Code Project pages Add shortcut Project settings

Create issue Import issues Configure fields Insights Create sprint

Project*: E-Commerce (COM)

Issue Type*: Story

Some issue types are unavailable due to incompatible field configuration and/or workflow associations.

Summary*: First time visitor can make the payment as a guest user without having to register

Description:
Style B I U A \approx A \approx \approx \approx \approx \approx
"As a first time visitor to the e-commerce website
-I want to be able to buy a listed product
-so that I can use the product I buy"

Assignee: Automatic

Assign to me

Labels

Sprint

Jira Software sprint field

Story point estimate

Measurement of complexity and/or size of a requirement.

Reporter*: Chintan Chovatiya

Start typing to get a list of possible matches.

Attachment: Drop files to attach, or browse.

Linked Issues: blocks

Issue

Flagged

Impediment

Allows to flag issues with impediments.

Create another **Create** Cancel Quickstart

Jira Software Your work Projects Filters Dashboards People Apps Create Search

E-Commerce Software project

Roadmap

Backlog

Board

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / E-Commerce

Backlog

Epic

Issues without epic

for a new e-commerce website to launch, the highest business value will be when a new user is able to buy an item from website.

+ Create Epic

COM Sprint 1 Add dates (1 issue)

COM-2 First time visitor can make the payment as a guest user without having to register

+ Create issue

Backlog (0 issues)

Your backlog is empty.

+ Create issue

Start sprint

Insights

Jira Software Your work Projects Filters Dashboards People Apps Create Search

E-Commerce Software project

Roadmap

Backlog

Board

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

Projects / E-Commerce

COM Sprint 1

9 days remaining Complete sprint ...

GROUP BY None Insights

TO DO 1 ISSUE

IN PROGRESS

DONE ✓

First time visitor can make the payment as a guest user without having to register

COM-2

Quickstart

434

Module – 3 [Context Based Testing]

Agenda

- Web Application Testing

- Usability Testing

- Compatibility Testing

- GUI Testing

- Security Testing

- Performance Testing

- Stress Testing

- Load Testing

- Desktop Application Testing

- Mobile Application Testing

Web Application Testing

Introduction

A **Web application** is an application that is accessed via Web browser over a network such as the Internet or an intranet. It is also a computer software application that is coded in a browser-supported language (such as HTML, JavaScript, Java, etc.)

In software engineering, a web application or web-application is an application that is accessed via a **web browser** over a network such as the Internet or an intranet.

The term may also mean a computer software application that is hosted in a browser-controlled environment (e.g. a Java applet) or coded in a browser supported language (such as JavaScript, possibly combined with a browser-rendered markup language like HTML) and reliant on a common web browser to render the application executable.

Web applications are popular due to the **ubiquity of web browsers**, and the convenience of using a **web browser as a client**, sometimes called a **thin client**.

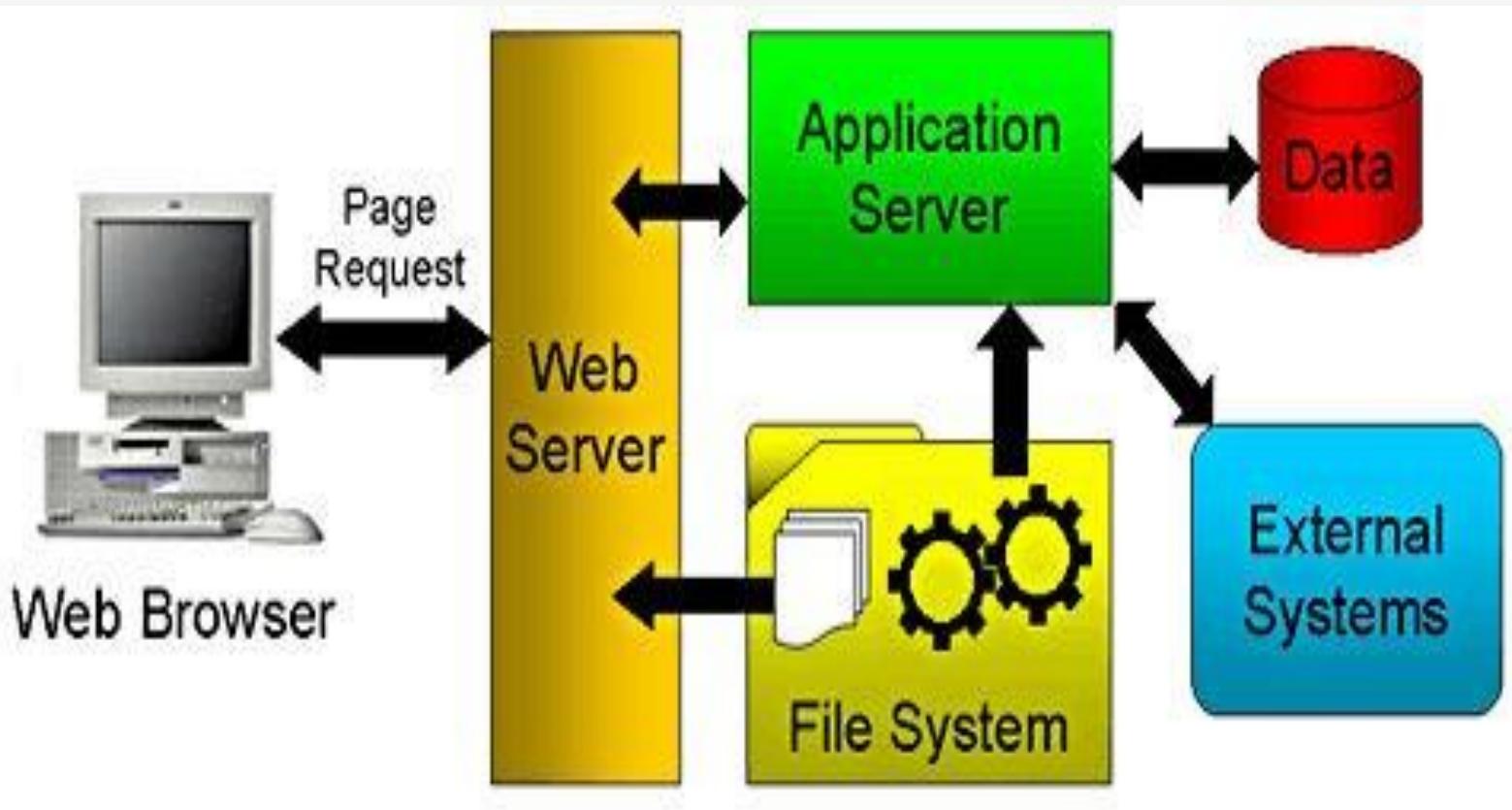
Introduction

- The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for **cross-platform compatibility**.
- Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.
- A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails or Struts2) is the middle tier (application logic), and a database is the third tier (storage).
- The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.
- Projects are broadly divided into two types of:
 - 2 tier applications
 - 3 tier applications

Client Server Testing

- This types of testing usually done for 2 tier applications (usually developed for LAN)
- Here we will be having front-end and backend.
- The application launched on front-end will be having forms and reports which will be monitoring and manipulating data
- E.g: applications developed in VB, VC++, Core Java, C, C++, Delphi, PowerBuilder etc.,
- The backend for these applications would be MS Access, SQL Server, Oracle, Sybase, Mysql, Quadbase

Client Server Architecture



Web Testing

- This is done for 3 tier applications (developed for Internet / intranet / xtranet)

- Here we will be having Browser, web server and DB server.

- The applications accessible in browser would be developed in HTML, DHTML, XML, JavaScript etc. (We can monitor through these applications)

- Applications for the web server would be developed in Java, ASP, JSP, VBScript, JavaScript, Perl, Cold Fusion, PHP etc. (All the manipulations are done on the web server with the help of these programs developed)

- The DB-Server would be having oracle, SQL server, Sybase, MySQL etc. (All data is stored in the database available on the DB server)

- Common Applications:

- Java
- PHP
- .NET
- Cold Fusion
- HTML, HHTML, DHTML
- XML, Java Script, VB Script and So on...

Web Testing Guide Lines

Functionality:

- **Links:** Objective is to check for all the links in the website.
 - All Internal Links
 - All External Links
 - All mail to links
 - Check for orphan Pages
 - Check for Broken Links
- **Forms:** Test for the integrity of submission of all forms.
 - All Field Level Checks
 - All Field Level Validations.
 - Functionality of Create, Modify, Delete & View.
 - Handling of Wrong inputs (Both client & Server)
 - Default Values if any
 - Optional versus Mandatory fields.
- **Cookies:** Check for the cookies that has to be enabled and how it has to be expired.

Web Testing Guide Lines

Functionality:

- **Web Indexing:** Depending on how the site is designed using Meta tags, frames, HTML syntax, dynamically created pages, passwords or different languages, our site will be searchable in different ways.
 - Meta Tags
 - Frames
 - HTML Syntax
- **Database:** Two types of errors that may occur in Web applications:
 - Data Integrity: Missing or wrong data in table.
 - Output Error: Errors in writing, editing or reading operations in the tables.

The issue is to test the functionality of the database, not the content and the focus here is therefore on output errors. Verify that queries, writing, retrieving or editing in the database is performed in a correct way.

Web Testing Guide Lines

Usability:

- **Navigation:** Navigation describes the way users navigate within a page, between different user interface controls (buttons, boxes, lists, windows etc.), or between pages via e.g. links.
 - Application navigation is proper through tab
 - Navigation through Mouse
 - Main features accessible from the main/home page.
 - Any hot keys, control keys to access menus.
- **Content:** Correctness is whether the information is truthful or contains misinformation. The accuracy of the information is whether it is without grammatical or spelling errors. Remove irrelevant information from your site. This may otherwise cause misunderstandings or confusion.
 - Spellings and Grammars
 - Updated information
- **General Appearance**
 - Page appearance
 - Color, font and size
 - Frames
 - Consistent design

Web Testing Guide Lines

Server Side Interfaces:

● Server Interface

- Verify that communication is done correctly, Web server-application server, application server-database server and vice versa.
- Compatibility of server software, hardware, network connections.
- Database compatibility (SQL, Oracle etc.)

● External Interface (if any)

Security:

- Valid and Invalid Login
- Limit defined for the number of tries.
- Can it be bypassed by typing URL to a page inside directly in the browser?
- Verify Log files are maintained to store the information for traceability.
- Verify encryption is done correctly if SSL is used (If applicable)
- No access to edit scripts on the server without authorization.

Web Testing Guide Lines

Client Side Compatibility:

- **Platform:** Check for the compatibility of
 - Windows (95, 98, 2000, NT)
 - Unix (different sets)
 - Macintosh (If applicable)
 - Linux
 - Solaris (If applicable)
- **Printing:** Despite the paperless society the web was to introduce, printing is done more than ever. Verify that pages are printable with considerations on:
 - Text and image alignment
 - Colours of text, foreground and background
 - Scalability to fit paper size
 - Tables and borders

Web Testing Guide Lines

Client Side Compatibility:

- **Browsers:** Check for the various combinations:
 - Internet Explorer (7.X 8.X, 9.X,10.X)
 - Netscape Navigator (6.X, 7.X, 8.X)
 - Goole Chrome (22.X.X)
 - Mozilla Fire Fox (11.X,12.X)
 - AOL
 - Browser settings (security settings, graphics, Java etc.)
- Frames and Cascade Style sheets
- Applets, ActiveX controls, DHTML, client side scripting
- HTML specifications.
- Graphics: Loading of images, graphics, etc.

Web Testing Guide Lines

Performance:

● Connection speed

- Try with Connection speed: 14.4, 28.8, 33.6, 56.6, ISDN, cable, DSL, T1, T3
- Time-out

● Load: Check/Measure the following:

- What is the estimated number of users per time period and how will it be divided over the period?
- Will there be peak loads and how will the system react?
- Can your site handle a large amount of users requesting a certain page?
- Large amount of data from users.

● Stress:

- Stress testing is done in order to actually break a site or a certain feature to determine how the system reacts. Stress tests are designed to push and test system limitations and determine whether the system recovers gracefully from crashes. Hackers often stress systems by providing loads of wrong in-data until it crash and then gain access to it during start-up.
 - Typical areas to test are forms, logins or other information transaction components.
 - Performance of memory, CPU, file handling etc.
 - Error in software, hardware, memory errors (leakage, overwrite or pointers)

Web Testing Guide Lines

Performance:

● Continuous use

- Is the application or certain features going to be used only during certain periods of time or will it be used continuously 24 hours a day 7 days a week?
- Will downtime be allowed or is that out of the question?
- Verify that the application is able to meet the requirements and does not run out of memory or disk space.

Testing Performed On Web

The tests performed on these types of applications would be

- Usability Testing
- GUI Testing means User Interface Testing
- Functionality Testing
- Security Testing
- Browser Compatibility Testing
- Performance Testing
- Load Testing
- Stress testing
- Interoperability Testing / Intersystem Testing
- Volume Testing means Storage and Data Volume Testing
- Database Testing means SQL Queries

Usability Testing

Need For Usability Testing

- Aesthetics and design are important. How well a product looks usually determines how well it works.
- There are many software applications / websites, which miserably fail, once launched, due to following reasons –

- Where do I click next?
- Which page needs to be navigated?
- Which Icon or Jargon represents what?
- Error messages are not consistent or effectively displayed
- Session time not sufficient.

Usability Testing identifies usability errors in the system early in development cycle and can save a product from failure.

Usability Testing Example

- **Web Based Testing , Desktop Based , Mobile based & Game based Testing :**

- All fields on a page (**For Example**, text box, radio options, drop-down lists) should be aligned properly.
- The user should not be able to type in drop-down select lists.
- Tab and Shift+Tab order should work properly.
- All buttons on a page should be accessible by keyboard shortcuts and the user should be able to perform all operations using a keyboard.
- All buttons on a page should be accessible by keyboard shortcuts and the user should be able to perform all operations using a keyboard...
- Check all pages for broken images.
- Check all pages for broken links.
- All pages should have a title.
- Confirmation messages should be displayed before performing any update or delete operation.
- Hourglass should be displayed when the application is busy.
- Page text should be left-justified.
- The user should be able to select only one radio option and any combination for checkboxes.

Goal of Usability Testing

- Effectiveness of the system
 - Efficiency
 - Accuracy
 - User Friendliness
-
- HOW MANY USERS DO YOU NEED?
 - 5(five) users are enough to uncover 80% of usability problems. Some researchers suggest other numbers. The truth is, the actual number of user required depends on the complexity of the given application and your usability goals. Increase in usability participant's results into increased cost, planning, participant management and data analysis.
 - But as a general guideline, if you on a small budget and interested in DIY usability testing 5 is a good number to start with. If budget is not a constraint its best consult experienced professionals to determine number of users.

Pros & Cons Usability Testing

Pros

- It helps uncover usability issues before the product is marketed.
- It helps improve end user satisfaction
- It makes your system highly effective and efficient
- It helps gather true feedback from your target audience who actually use your system during usability test. You do not need to rely on “opinions” from random people.

Cons

- Cost is a major consideration in usability testing. It takes lots of resources to set up a Usability Test Lab. Recruiting and management of usability testers can also be expensive
- However, these costs pay themselves up in form of higher customer satisfaction, retention and repeat business. Usability testing is therefore highly recommended.

Compatibility Testing

Introduction

- In computer world, compatibility is to check whether your software is capable of running on different hardware, operating systems, applications, network environments or mobile devices.

- Compatibility Testing is a type of the **Non-functional testing**.

Types of Compatibility Testing

- Hardware
- Operating Systems
- Software
- Network
- Browser
- Devices
- Mobile
- Versions of the software
 - Backward compatibility Testing
 - Forward compatibility Testing



Compatibility Testing

Tools

Adobe Browser Lab – Browser Compatibility Testing:

- This tool helps us to check your application in different browsers.

Secure Platform – Hardware Compatibility tool

- These tools include necessary drivers for a specific hardware platform and it provides information on tool to check for CD burning process with CD burning tools.

Virtual Desktops – Operating System Compatibility:

- This is used to run the applications in multiple operating systems as virtual machines.
- Number of systems can be connected and compare the results.

Compatibility Testing Example

- **Web Based Testing & Desktop Based Testing :**
 - The application is compatible with different sizes such as RAM, hard disk, processor, and the graphic card, etc.
 - Check that the application is compatible with mobile platforms such as iOS, Android, etc.
 - Checking the compatibility of the software in the different network parameters such as operating speed, bandwidth, and capacity.
- **Mobile Based Testing :**
 - Define the platforms on which mobile app is likely to be used
 - Create the device compatibility library
 - Make a drawing of various environments, their hardware's, and software to figure out the behavior of the application in different configurations
 - Again perform the testing by following the same process, till no bugs can be found.
- **Game Based Testing :**
 - Validate whether the user interface of the app is as per the screen size of the device and ensure high quality
 - Ensures real compatibility between different testing environments with android phone and ipone

GUI Testing

Introduction

Graphical User Interface (GUI) testing is the process of testing the system's GUI of the System under Test. GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars – tool bar, menu bar, dialog boxes and windows etc.

WHAT DO YOU CHECK IN GUI TESTING?

- Check all the GUI elements for size, position, width, length and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
- Check you can execute the intended functionality of the application using the GUI
- Check Error Messages are displayed correctly
- Check for Clear demarcation of different sections on screen
- Check Font used in application is readable
- Check the alignment of the text is proper
- Check the Color of the font and warning messages is aesthetically pleasing
- Check that the images have good clarity
- Check that the images are properly aligned
- Check the positioning of GUI elements for different screen resolution.

Approach of GUI Testing

MANUAL BASED TESTING

Under this approach, graphical screens are checked manually by testers in conformance with the requirements stated in business requirements document.

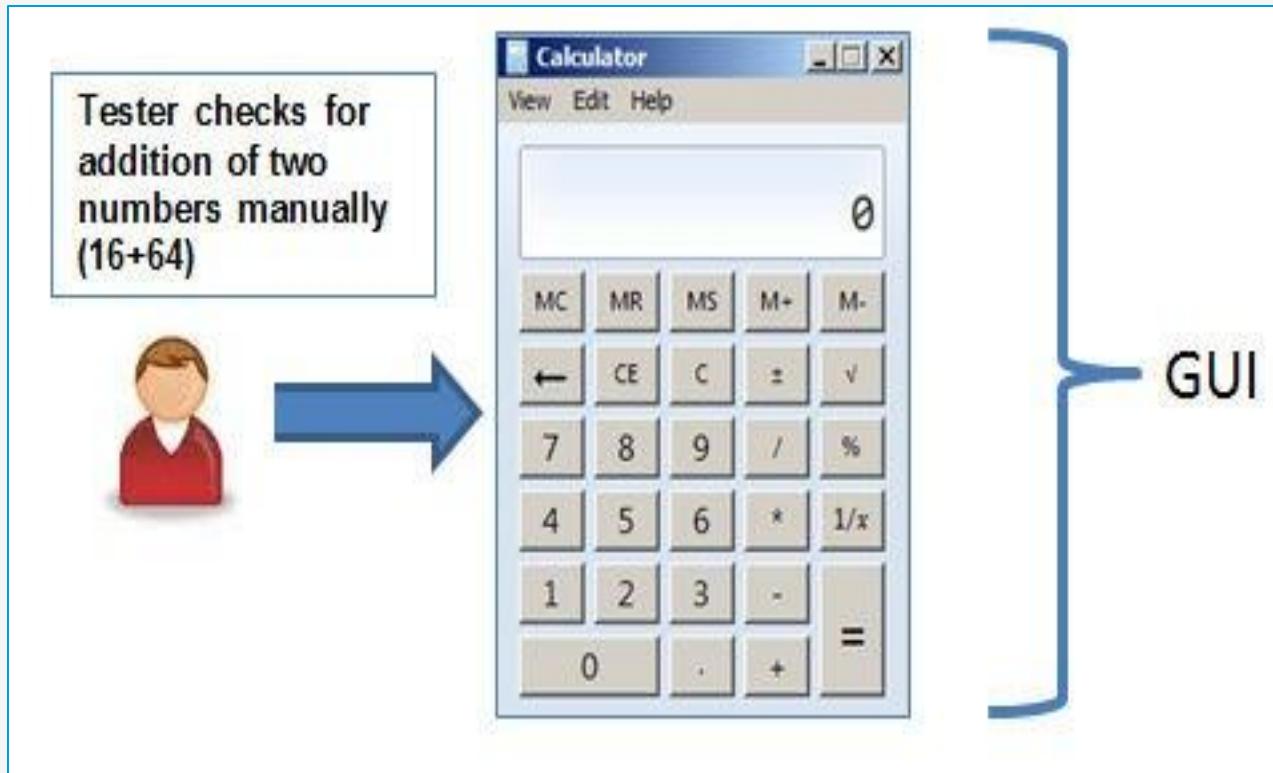
RECORD AND REPLAY

GUI testing can be done using automation tools. This is done in 2 parts. During Record , test steps are captured into the automation tool. During playback, the recorded test steps are executed on the Application under Test. Example of such tools - QTP.

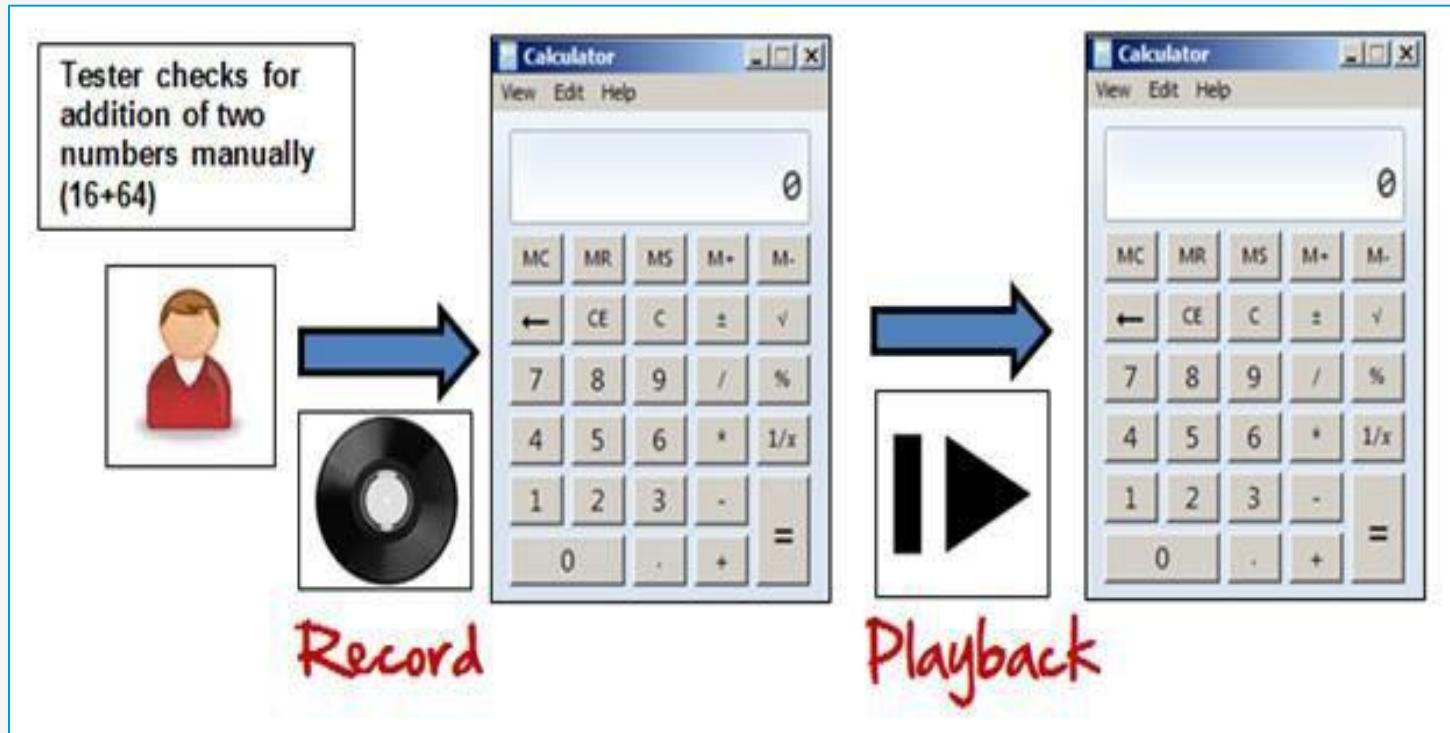
MODEL BASED TESTING

A model is a graphical description of system's behavior. It helps us to understand and predict the system behavior. Models help in a generation of efficient test cases using the system requirements.

Manual Based Testing in GUI



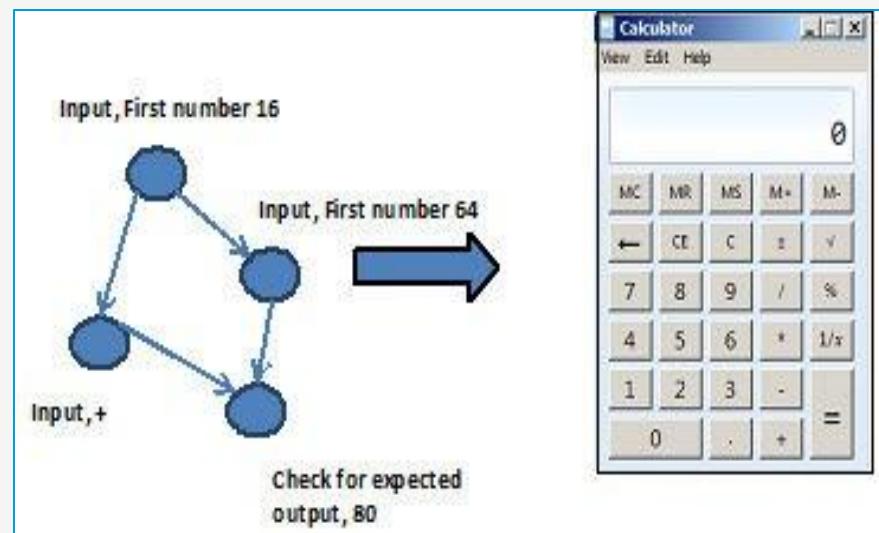
Record & Play in GUI



Model Based Testing in GUI

- Build the model
- Determine Inputs for the model
- Calculate expected output for the model
- Run the Tests
- Compare the actual output with the expected output
- Decision on further action on the model
- Some of the modeling techniques from which test cases can be derived:
 - Charts – Depicts the state of a system and checks the state after some input.
 - Decision Tables – Tables used to determine results for each input applied

- Model based Testing is an evolving technique for the generating the test cases from the requirements.
- Its main advantage, compared to above two methods, is that it can determine undesirable states that your GUI can attain.



GUI Testing Examples

- **Web Based Testing & Desktop Based Testing :**
 - The scrollbar should be enabled only when necessary.
 - Font size, style, and color for headline, description text, labels, infield data, and grid info should be standard as specified in SRS.
 - The description text box should be multi-lined.
 - Enough space should be provided between field labels, columns, rows, error messages, etc.
- **Mobile Based Testing :**
 - If mobile is in every orientation mode so display image , video properly.
 - Every app will display in responsive type .
 - Alignment should be apply properly of every field.
- **Game Based Testing :**
 - Game infra design will showing properly
 - Game points or score will display proper with its background color.
 - Game sound manage with its background effect
 - Can be also conducted in advance of designing page layouts or navigation menus

Security Testing

Introduction

- You need to test how secure your web application is from both external and internal threats.
- The security of your web application should be planned for and verified by qualified security specialists.
- Security is set of measures to protect an application against unforeseen actions that cause it to stop functioning or being exploited.
- Security Testing ensures that system and applications in an organization are free from any loopholes that may cause a big loss.
- Security testing of any system is about finding all possible loopholes and weaknesses of the system which might result into loss of information at the hands of the employees or outsiders of the Organization.
- **The goal of security testing is to identify the threats in the system and measure its potential vulnerabilities.**
- It also helps in detecting all possible security risks in the system and help developers in fixing these problems through coding.

Types of Security Testing

- **Vulnerability Scanning**
- **Security Scanning**
- **Risk Assessment**
- **Security Auditing**
- **Ethical hacking**
- **Posture Assessment**

ROLES YOU MUST KNOW!

- **Hackers** - Access computer system or network without authorization
- **Crackers** – Break into the systems to steal or destroy data
- **Ethical Hacker** – Performs most of the breaking activities but with permission from owner
- **Script Kiddies or packet monkeys** – Inexperienced Hackers with programming language skill

Security with SDLC

SDLC Phases	Security Processes
Requirements	Security analysis for requirements and check abuse/misuse cases
Design	Security risk analysis for designing. Development of test plan including security tests
Coding and Unit Testing	Static and Dynamic Testing and Security white box testing
Integration Testing	Black Box Testing
System Testing	Black Box Testing and Vulnerability scanning
Implementation	Penetration Testing, Vulnerability Scanning
Support	Impact analysis of Patches

Security Testing Techniques

- In security testing, different methodologies are followed, and they are as follows:
 - **Tiger Box:** This hacking is usually done on a laptop which has a collection of OSs and hacking tools. This testing helps penetration testers and security testers to conduct vulnerabilities assessment and attacks.
 - **Black Box:** Tester is authorized to do testing on everything about the
 - network topology and the technology.
 - **Grey Box:** Partial information is given to the tester about the system, and it is hybrid of white and black box models.

Security Testing Examples

- **Web Based Testing & Desktop Based Testing :**
 - Secure pages should use the HTTPS protocol.
 - Sensitive fields like passwords and credit card information should not have to autocomplete enabled.
 - Verify CAPTCHA functionality.
- **Mobile Based Testing :**
 - Every payment gateway app used security code or OTP
 - Must be used mobile tracking mode on
- **Game Based Testing :**
 - Secure your rewards which you get from your game playing
 - Without current round completing , you can not going in next round.

Performance Testing

Introduction

Software performance testing is a means of quality assurance (QA). It involves testing software applications to ensure they will perform well under their expected workload.

Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, do matter. The goal of performance testing is not to find bugs but to eliminate performance bottlenecks

The focus of Performance testing is checking a software programs

- **Speed** – Determines whether the application responds quickly
- **Scalability** – Determines maximum user load the software application can handle.
- **Stability** – Determines if the application is stable under varying loads

Types of Performance Testing

- Load testing
- Stress testing
- Endurance testing
- Spike testing
- Volume testing
- Scalability testing

Performance Problems

Long Load time -

- Load time is normally the initial time it takes an application to start.
- This should generally be kept to a minimum.
- While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.

Poor response time -

- Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input.
- Generally this should be very quick.
- Again if a user has to wait too long, they lose interest.

Poor scalability -

- A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users.
- Load testing should be done to be certain the application can handle the anticipated number of users.

Performance Problems

Bottlenecking –

- Bottlenecks are obstructions in system which degrade overall system performance.
- Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads.
- **Bottlenecking is often caused by one faulty section of code.**
- The key to fixing a bottlenecking issue is to find the section of code that is causing the slowdown and try to fix it there.
- Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware.
- Some **common performance bottlenecks** are
 - CPU utilization
 - Memory utilization
 - Network utilization
 - Operating System limitations
 - Disk usage

Performance Parameters

Processor Usage

- Bandwidth
- Private bytes
- Committed memory
- Memory pages/second
- Page faults/second
- CPU interrupts per second
- Disk queue length
- Network output queue length
- Network bytes total per second
- Response time

- Throughput
- Amount of connection pooling
- Maximum active sessions
- Hit ratios
- Hits per second
- Rollback segment
- Database locks
- Top waits
- Thread counts
- Garbage collection

Performance Test Tools

HP Load runner –

- is the most popular performance testing tools on the market today.
- This tool is capable of simulating hundreds of thousands of users, putting applications under real life loads to determine their behavior under expected loads.
- Load runner features a virtual user generator which simulates the actions of live human users.

HTTP Load –

- a throughput testing tool aimed at testing web servers by running several http or https fetches simultaneously to determine how a server handles the workload.

Proxy Sniffer –

- one of the leading tools used for load testing of web and application servers.
- It is a cloud based tool that's capable of simulating thousands of users.

Performance Testing Examples

- **Web Based Testing & Desktop Based Testing :**
 - Check the page load on slow connections.
 - Check the database query execution time.
 - Check the performance of database stored procedures and triggers.
- **Mobile Based Testing :**
 - Check the database query execution time.
 - Check the response time for any action under a light, normal, moderate, and heavy load conditions.
 - Check CPU and memory usage under peak load conditions
- **Game Based Testing :**
 - Determine whether the current infrastructure is sufficient for the smooth running of the game
 - Determine the number of users an app can support and its scalability rate to support more users
 - Accommodates strategies for performance management.

Stress Testing

Introduction

- **Stress testing** - System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to system or database load.
- Stress testing is used to test the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions.
- **It even tests beyond the normal operating point and evaluates how the system works under those extreme conditions.**
- **Stress Testing is done to make sure that the system would not crash under crunch situations.**
- **Stress testing is also known as endurance testing.**

Introduction

- Under Stress Testing, AUT is stressed for a short period of time to know its withstanding capacity.
- Most prominent use **of stress testing is to determine the limit, at which the system or software or hardware breaks.**
- It also checks whether system demonstrates effective error management under extreme conditions.
- The application under testing will be stressed when 5GB data is copied from the website and pasted in notepad.
- Notepad is under stress and gives ‘Not Responded’ error message.
- Examples of stress conditions include:
 - Excessive volume in terms of either users or data; examples might include a denial of service (DoS) attack or a situation where a widely viewed news item prompts a large number of users to visit a Web site during a three-minute period.
 - Resource reduction such as a disk drive failure.
 - Application components fail to respond.

Need For Stress Testing

- During festival time, an online shopping site may witness a spike in traffic, or when it announces a sale.
- When a blog is mentioned in a leading newspaper, it experiences a sudden surge in traffic.
- To check whether the system works under abnormal conditions.
- Displaying appropriate error message when the system is under stress.
- System failure under extreme conditions could result in enormous revenue loss
- It is better to be prepared for extreme conditions by executing Stress Testing.

Goal of Stress Testing

- The goal of stress testing is to analyze the behavior of the system after failure. For stress testing to be successful, system should display appropriate error message while it is under extreme conditions.
- To conduct Stress Testing, sometimes, massive data sets may be used which may get lost during Stress Testing.
- Testers should not lose this security related data while doing stress testing.
- The main purpose of stress testing is to make sure that the system recovers after failure which is called as **recoverability**.

Types of Stress Testing

- Application Stress Testing:
- Transactional Stress Testing:
- Systemic Stress Testing:
- Exploratory Stress Testing:

Stress Testing

- Stress Tester
- Neo Load
- App Perfect

Metrics for Stress Testing

Measuring Scalability & Performance

- **Pages per Second:** Measures how many pages have been requested / Second
- **Throughput:** Basic Metric – Response data size/Second
- **Rounds:** Number of times test scenarios has been planned Versus Number of times client has executed

Application Response

- **Hit time:** Average time to retrieve an image or a page
- **Time to the first byte:** Time taken to return the first byte of data or information
- **Page Time:** Time taken to retrieve all the information in a page

Failures

- **Failed Connections:** Number of failed connections refused by the client
- **Failed Rounds:** Number of rounds it gets failed
- **Failed Hits:** Number of failed attempts done by the system

Load Testing

Introduction

Load testing - Its a performance testing to check system behavior under load. Testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.

Load testing is a kind of performance testing which determines a system's performance under real-life load conditions. This testing helps determine how the application behaves when multiple users access it simultaneously.

This testing usually identifies –

- The maximum operating capacity of an application
- Determine whether current infrastructure is sufficient to run the application
- Sustainability of application with respect to peak user load
- Number of concurrent users that an application can support, and scalability to allow more users to access it.
- It is a type of non-functional testing. Load testing is commonly used for the Client/Server, Web based applications – both Intranet and Internet.

Need For Load Testing

Some extremely popular sites have suffered serious downtimes when they get massive traffic volumes. E-commerce websites invest heavily in advertising campaigns, but not in Load Testing to ensure optimal system performance, when that marketing brings in traffic.

For Example

- Popular toy store **Toysrus.com**, could not handle the increased traffic generated by their advertising campaign resulting in loss of both marketing dollars, and potential toy sales.
- An Airline website was not able to handle 10000+ users during a festival offer.
- Encyclopedia Britannica declared free access to their online database as a promotional offer. They were not able to keep up with the onslaught of traffic for weeks.
- Facebook(FB)

Why Load Testing?

- Load testing gives confidence in the system & its reliability and performance.
- Load Testing helps identify the bottlenecks in the system under heavy user stress scenarios before they happen in a production environment.
- Load testing gives excellent protection against poor performance and accommodates complementary strategies for performance management and monitoring of a production environment.

Goals of Load Testing

● Loading testing identifies the following problems before moving the application to market or Production:

- Response time for each transaction
- Performance of System components under various loads
- Performance of Database components under different loads
- Network delay between the client and the server
- Software design issues
- Server configuration issues like Web server, application server, database server etc.
- Hardware limitation issues like CPU maximization, memory limitations, network bottleneck, etc.

● Load testing will determine whether system needs to be fine-tuned or modification of hardware and software is required to improve performance.

Pre-Requisites for Load Testing

The chief metric for load testing is response time. Before you begin load testing, you must determine -

- Whether the response time is already measured and compared – Quantitative
- Whether the response time is applicable to the business process – Relevant
- Whether the response time is justifiable – Realistic
- Whether the response time is achievable – Achievable
- Whether the response time is measurable using a tool or stopwatch – Measurable

Hardware Platform Software Configuration

- | | |
|--|--|
| <ul style="list-style-type: none">· Server Machines· Processors· Memory· Disk Storage· Load Machines configuration· Network configuration | <ul style="list-style-type: none">· Operating System· Server Software |
|--|--|

Strategies of Load Testing

- Manual Load Testing
- In house(Organization) developed load testing tools
- Open source load testing tools
- Enterprise(Record and Play) load testing tools

Load Testing

- Loadrunner
- Web Load
- Astra Load Test
- Review's Web Load
- Studio, Rational Site Load
- Silk Performer

Pros and Cons of Load Testing

Pros:

- Performance bottlenecks identification before production
- Improves the scalability of the system
- Minimize risk related to system down time
- Reduced costs of failure
- Increase customer satisfaction

Cons:

- Need programming knowledge to use load testing tools.
- Tools can be expensive as pricing depends on the number of virtual users supported.

Load Testing vs Stress Testing

Load Testing Stress Testing

Load Testing is to test the system behavior under normal workload conditions, and it is just testing or simulating with the actual workload.

Load testing identifies the bottlenecks in Stress testing determines the breaking the system under various workloads and point of the system to reveal the maximum checks how the system reacts when the point after which it breaks. load is gradually increased

Load testing does not break the system

Stress testing is to test the system behavior under extreme conditions and is carried out till the system failure.

Stress testing tries to break the system by testing with overwhelming data or resources.

Desktop Application Testing

Desktop Testing Guide Lines

The tests performed on these types of applications would be

- User interface testing
- Manual support testing
- Functionality testing
- Compatibility testing
- Configuration testing
- Intersystem testing

● Application runs in single memory (Front end and Back end in one place)

● Single user only

Mobile Application Testing

Introduction

- The below checklist ensures that both developers and testers have covered these high level scenarios during their requirements discussion, development and testing activities.

- Mobile application development and testing checklist** also helps you refine your requirements to ensure that your scope of work is clearly defined.

- These are high level questions and not very specific to the application functionality



iPhone



iPad



Android



Windows

Challenges

- Unique challenges with mobile applications; device compatibility, OS compatibility, UI compatibility, browser compatibility (mobile web apps), network, etc.
- Closed Systems –Device, OS, network, etc. are specified; controlled platform and straight forward testing strategy
- Open Systems –Application can be installed across a variety of devices, OSs, networks, etc.
- Many combinations and the most challenging testing strategy many device types, rapid upgrading of OSs, more than 40 mobile browsers, over 400 network operators

Challenges

- For Open Systems, different techniques and methods can be used in testing
- Understand the OS, device and network landscape for the intended user base of the application before testing
- Conduct testing in an uncontrolled, real-word environment, as well as in a controlled, simulated environment
- An optimal strategy will combine different testing options both manual and automated, that together provide the best overall result to balance tradeoffs between cost and time-to-market, while assuring the highest quality

Android Mobile Version

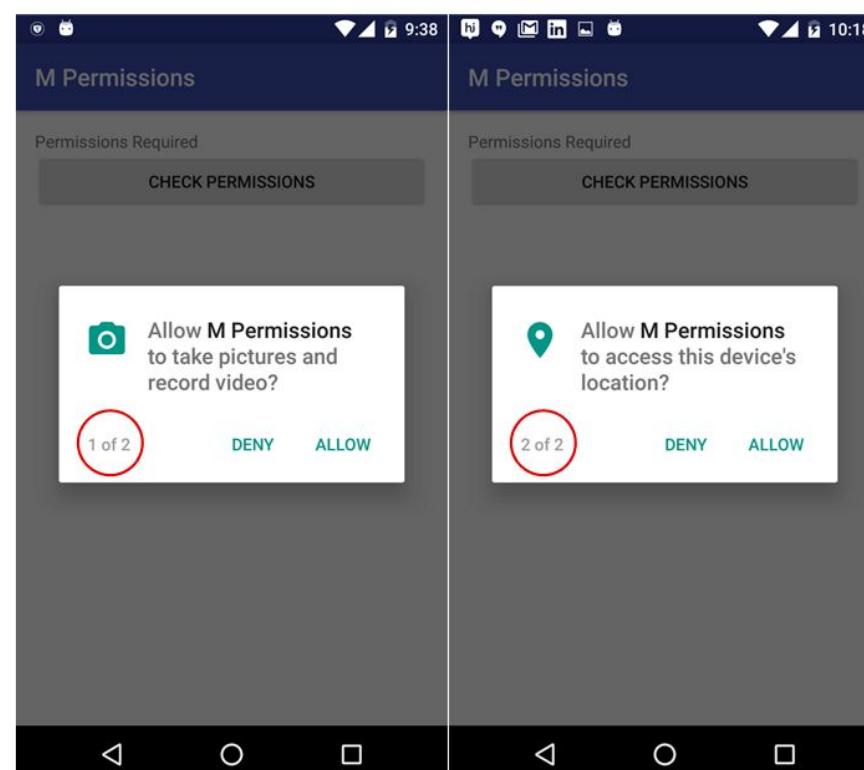
Name	Version number(s)	Initial stable release date	Supported (security fixes)	API level	References
No official codename	1.0	September 23, 2008	No	1	[9]
	1.1	February 9, 2009	No	2	[9][14]
Cupcake	1.5	April 27, 2009	No	3	[15]
Donut	1.6	September 15, 2009	No	4	[16]
Eclair	2.0 – 2.1	October 26, 2009	No	5 – 7	[17]
Froyo	2.2 – 2.2.3	May 20, 2010	No	8	[18]
Gingerbread	2.3 – 2.3.7	December 6, 2010	No	9 – 10	[19]
Honeycomb	3.0 – 3.2.6	February 22, 2011	No	11 – 13	[20]
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011	No	14 – 15	[21]
Jelly Bean	4.1 – 4.3.1	July 9, 2012	No	16 – 18	[22]
KitKat	4.4 – 4.4.4	October 31, 2013	No	19 – 20	[23]
Lollipop	5.0 – 5.1.1	November 12, 2014	No	21 – 22	[24]
Marshmallow	6.0 – 6.0.1	October 5, 2015	No	23	[25]
Nougat	7.0 – 7.1.2	August 22, 2016	No	24 – 25	[26][27][28][29]
Oreo	8.0 – 8.1	August 21, 2017	Yes	26 – 27	[30]
Pie	9	August 6, 2018	Yes	28	[31]
Android 10	10	September 3, 2019	Yes	29	[32]
Android 11	11	September 8, 2020	Yes	30	[33]

iPhone versions

iPhone SE (1st)	iOS 9.3	March 31, 2016	September 12, 2018
iPhone 7 / 7 Plus	iOS 10.0.1	September 16, 2016	September 10, 2019
iPhone 8 / 8 Plus	iOS 11.0	September 22, 2017	April 15, 2020
iPhone X	iOS 11.0.1	November 3, 2017	September 12, 2018
iPhone XS / XS Max	iOS 12.0	September 21, 2018	September 10, 2019
iPhone XR	iOS 12.0	October 26, 2018	
iPhone 11	iOS 13.0	September 20, 2019	
iPhone 11 Pro / 11 Pro Max	iOS 13.0	September 20, 2019	October 13, 2020
iPhone SE (2nd)	iOS 13.4	April 24, 2020	
iPhone 12 / 12 Mini	iOS 14.1 (12) / iOS 14.2 (12 Mini)	October 23, 2020 (12) / November 13, 2020 (12 Mini)	
iPhone 12 Pro / 12 Pro Max	iOS 14.1 (12 Pro) / iOS 14.2 (12 Pro Max)	October 23, 2020 (12 Pro) / November 13, 2020 (12 Pro Max)	

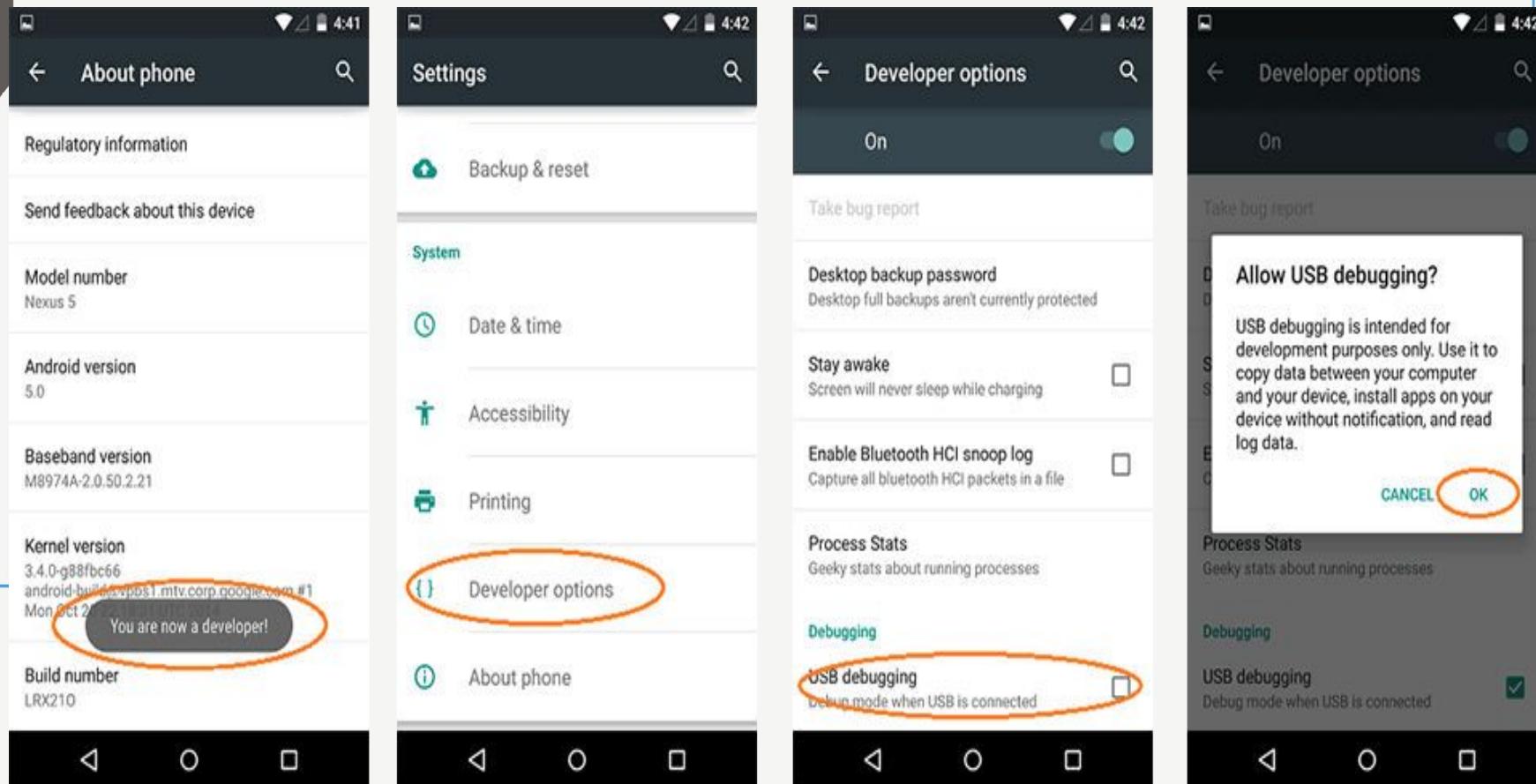
Android permission

Android M Permissions -Requesting Multiple permissions



www.androidhive.info

Developer option and USB ON



Guide Lines for Mobile Testing

Which Mobile Platform To Develop For? Ios Or Android?

- iOS and Android are the preferred platform for developing mobile applications.
- However, Blackberry is still used by several enterprise users and a significant population of the developing world still uses Symbian phones.
- It's good to know upfront which platforms are expected to be supported.
- This will help you make crucial decisions regarding your architecture / design and also provide inputs on whether you want to develop a completely native application or use a framework like Phone Gap.
- Popular platforms are listed below:
 - Android
 - iOS
 - Windows Mobile
 - Blackberry
 - Symbian

Guide Lines for Mobile Testing

Which Version Of Ios Should I Target? What Version Of Android Should I Develop For?

- Does your application use any features introduced in a specific version of the OS?
- If so, you will want to mention this in your marketing material and also test it on the target OS.
- It is also good practice to prevent the application from being installed on OS versions that are not supported.
- This will avoid users leaving low ratings and negative feedback after installing the application on an unsupported OS.
- Of course, you will want to test and ensure that your application works on the targeted OS version.

Guide Lines for Mobile Testing

Device Hardware Requirements

- Does your application have any specific hardware requirements like memory, camera, CPUs etc? As mentioned above, it's best to prevent installation on unsupported devices programmatically when possible. Here are instructions to check if your device has sufficient memory/RAM in Android and iOS and also to check for camera on iOS and Android.

Which Screen Resolution Should I Target?

- Make sure that your application looks good on your target screen resolution. Smart phones and tablets come in all shapes and sizes. A list of devices with screen resolution and display density is available on Wikipedia. Some of the common screen resolutions are below:

- 320 x 480px
- 640 x 960px
- 480 x 800px
- 720 x 1280px
- 768 x 1280px
- 800 x 1280px
- 1200 x 1920px
- 2048 x 1536px

Guide Lines for Mobile Testing

Should I Develop Another App For Tablets?

- It's good practice to use **high quality graphics for large devices** like tablets especially if your application or game is expected to be used on these devices.
- Some developers release a **separate HD version of the application/game** instead of using a single package.

Portrait Or Landscape Orientation?

- Some games work only in landscape mode while some applications are designed to work in portrait mode only and other work in both modes.
- Make sure you test your applications to see if there are any issues when changing the orientation like application crashing or UI bugs.

Test Mobile + Web App Updates

- Does your mobile application have a server side component or a web service it uses?
- Does the mobile application need an update when the server side component is updated?
- If so, make sure there is a test case to track this to avoid any human error.

Guide Lines for Mobile Testing

Testing GPS Functionality, Accelerometer, Hardware Keys

If your application requires use of the following hardware features, your test cases also need to test for scenarios when they are not available -

- Hardware keys – Ex. Camera application using a dedicated camera button, Task/Event Manager applications using hardware buttons to snooze a reminder, media players using volume and other keys etc. Some applications also use the power button to provide additional functionality / shortcuts to application behavior.
- Accelerometer – Applications that make use of accelerometer require testing to ensure that the readings are being recorded accurately and utilized correctly within the applications. This test case might be relevant to applications like Star Maps, Pedometers, Jump trackers, Games, 3D visualization applications etc
- GPS – How will your Navigation applications respond if the GPS is disabled or turned off abruptly during operation?
- Any other sensor – If your application depends on additional sensors for temperature, luminosity or any accessory that provides additional functionality, then you need to ensure that you have tested for conditions when they are not available or do not function accurately.

Guide Lines for Mobile Testing

Network Connectivity Issues – GPRS, 2g, 3g, Wi-Fi, Intermittent Connectivity, No Connectivity

- Most of the applications are developed in the presence of Wi-Fi which provides good network connectivity.
- However it's important to test applications in the real world where the user might not have access to WiFi.
- Usually when people are on the move, network connectivity is intermittent with connection being dropped once in a while.
- Network speeds also vary based on the users location and the kind of connectivity they are paying for.
- Applications must be able to handle these situations gracefully and they must be tested for it.

Guide Lines for Mobile Testing

Testing Interruptions To The Mobile App

There are various events that can interrupt the flow of your application. Your application should be able to handle these and should be tested for the same.

- Incoming Call
- Text message
- Other app notifications
- Storage low
- Battery low
- Battery dead
- No storage
- Airplane mode
- Intermittent connectivity
- Home screen jump
- Sleep mode

Guide Lines for Mobile Testing

Mobile Application Security Testing

Security and data privacy are of utmost importance in today's scenario. Users are worried about their data and credentials being exposed through vulnerable applications.

- Is your application storing payment information or credit card details?
- Does your application use secure network protocols?
- Can they be switched to insecure ones?
- Does the application ask for more permissions than it needs?
- Does your application use certificates?
- Does your application use a Device ID as an identifier?
- Does your application require a user to be authenticated before they are allowed to access their data?
- Is there a maximum number of login attempts before they are locked out?
- Applications should encrypt user name and passwords when authenticating the user over a network. One way to test security related scenarios is to route your mobile's data through a proxy server like OWASP Zed Attack Proxy and look for vulnerabilities.

Guide Lines for Mobile Testing

Testing In-App Payment, Advertisements And Payment Gateway Integrations

- If your app makes use of in-app payment, advertisements or payment gateways for e-commerce transactions, you will need to test the functionality end to end to ensure that there are no issues in the transactions.
- Testing for payment gateway integration and advertisements will need accounts to be created with the Payment Gateways and Advertisement servers before testing can begin.

Testing Social Network Integration

- Many applications these days ship with the ability to share a post from the application, on the users' social networking account.
- However most users would like to be prompted before a post is published on their account. Does your application handle this?
- Are they being allowed to share the status message being shared?

Guide Lines for Mobile Testing

Mobile Application Performance Testing

- Have you checked to see if the performance of your mobile application degrades with increase in the – size of mailbox, album, messages, music or any other content relevant to the application?
- It's good practice to test your application for performance and scalability issues.
- With large storage capacity being available at affordable prices, it's not uncommon for users to have large amount of data / content on their smart phone.
- Users even store SMS for several years on their smart phones.
- If your application has user generated content / data associated with it (Ex. Photographs, SMS etc) which can grow to huge proportions over the lifetime of the application, your testing should include these scenarios to see how the application performs.
- In case the application has a server side component, you should also test the application with increasing number of users.
- While this testing can be done manually, we have tools like Little Eye and Neo Load that can help you with Performance and Load testing of your mobile application.

Guide Lines for Mobile Testing

Mobile Application Localization And Time Zone Issues

- If your application is multilingual, it needs to be tested in other languages to ensure that there is no character encoding issue, data truncation issue or any UI issues due to varying character lengths.
- You also need to test applications to ensure that they handle time zone changes.
- What happens if a user travels forward across time zone and returns to his/her previous time zone?
- How does your app handle entries with date and time which are in sequence but not in chronological order?

Guide Lines for Mobile Testing

Test Hardware Connectivity – Bluetooth, Wi-Fi, Nfc, Usb – Device Recognition

- Smart phones come with a plethora of connectivity options. If your application makes use of the below connectivity options (Ex. File managers or photo editors that let you share your file, Air Droid which allows you to transfer files between PC and your mobile over wi-fi) then you should test them to ensure they work as expected.
- You should also test to see how they handle errors when the connection is lost during a transfer / transaction.
- Commonly used mechanism to share data or transact are:
 - Bluetooth
 - Wi-Fi
 - USB
 - NFC

Guide Lines for Mobile Testing

Google Play / Apple App Store Integration And Supported Device List/Restrictions

- Consultants and organizations that provide end to end services should also include test cases to ensure that the mobile app is successfully deployed to the App store / Play store and is only available to the supported devices.
- This could also include validation of all the text, screenshots, and version numbers etc that are part of the app listing.

Mobile Testing Process

Test Strategy

- a high level document that defines “Testing Approach” to achieve testing objectives. Components of the document include: Approach, Risks, Contingencies & Recommendations, Testing Matrix, Defect Management Process and Resource Requirements (schedule, tools, roles & responsibilities).

Test Plan

Test Design Specification

- outlines, defines and details the approach taken to perform mobile application testing. The objective is to identify user flows and annotations, features to be tested, test scenarios, acceptance and release criteria. Depending on the project, can be combined with Test Plan.

Test Cases

- are derived from test scenarios identified in the Test Design Specification. They are a set of test actions, test data/user input data, execution conditions, and expected results developed to verify successful and acceptable implementation of the application requirements.

Mobile Testing Process

Test Case Execution Summary Report

- Report provides information uncovered by the tests. The report is used to document the overall status of test execution including: proof of test execution (FDA regulated applications), test pass or fail status, defects identified, resolution of defects and any remaining open defects, regression testing results and overall testing effort conclusions.

Mobile Testing Techniques

- Manual and Automated Testing
- Database Testing
- Compatibility Testing
- Functional Testing
- Interoperability Testing
- Mobile Analytics Testing
- Power Consumption Testing
- Usability Testing
- Security Testing
- Mobile Device Emulators

Mobile Testing Techniques

Manual Testing

- Test scripts are executed by a tester using an actual device under various scenarios
- The most labor intensive and time consuming technique but necessary for select, critical test cases

Automated Testing

- Test scripts are executed using emulator(s) and performance testing tool(s) eggPlant is an example of a broad based QA automation testing tool that emulates mobile devices and automates the testing; eggPlant can be downloaded for the Windows or Mac platforms; interfaces with devices, device emulators and other tools to consolidate/complete the testing process

Mobile Testing Techniques

Examples of Mobile Application Test Automation Tools

eggPlant	Robitium	Fone Monkey	Windows Mobile
eggPlant Server & Console ,Sensetalk Scripting	Robitium.Solo.jar 3.2	FoneMonkey Android FoneMonkey IOS5.5b FoneMonkey Console	Visual Studio 2010 Express
Mobile VNC Viewer & VNC Server	Eclipse 3.7	Eclipse, Xcode, AspectJ development Tool.	WP SDK
iPhone Simulator, Android Emulator, WP Emulator	Android Emulator 4.0.3	iPhone Simulator Android Emulator WP Emulator	Windows Phone Emulator
Android and iOS Devices, Windows Phone	Android Device	Android and iOS Devices	Windows Phone

Mobile Testing Techniques

Database Testing

- Check for data integrity and errors while editing, deleting and modifying the forms and all other DB related information
- Executed manually without use of tools

Compatibility Testing

- Assures the application works as intended with the selected device, operating system, screen size, display and internal hardware
- Can be tested with automation, the following are example tools that simulate different devices, operating systems, screens, etc.
 - iPHoney, a free iPhone simulator powered by Safari (used on a MAC OS platform only); iPadPeek, a free iPadsimulator
 - Adobe Device Central CS5, supports the preview, test and delivery of mobile applications; available with the Adobe Creative Suite editions: Photoshop, Illustrator, Flash Professional, Dreamweaver After Effects and Fireworks.
 - DeviceAnywhere, supports automated testing that runs across multiple devices and multiple platforms/OS's
- Use of manual testing and actual devices for the most critical systems is recommended

Mobile Testing Techniques

Functional Testing

- Includes the testing of controls, storage media handling options, and other operational aspects
- Functionality testing for the mobile application is black-box testing and assures that the application functions per the business specifications
- Executed manually without use of tools

Interoperability Testing

- Includes the testing of different functionalities within the device, for example;
- Does iTunes end the play of music when launching the application under test?
- What happens when a call is received and the application is being launched?
- Can be tested with a combination of manual and automated test methods, but primarily executed using manual tests

Usability Testing

- Used to verify mobile interface (UI), navigation, and application, as well as consistency, and soberness
- Executed manually without use of tools

Mobile Testing Techniques

Mobile Analytics Testing

- Test the correct implementation of analytics: verify page and link tags, redirects, page source and user attributes as well as data capture
- Can be tested with automation, the following are example tools:
 - Charles Web Debugging Proxy to verify the page and link tags
 - Flurry Dashboard to validate the data was captured correctly, the dashboard view provides a snapshot of user metrics and usage
 - Performance Testing is used to load and stress test the mobile application and database servers; Empirixe Tester to perform load and stress testing, eLoadExpert simulates concurrent users
 - Test performance under realistic conditions of wireless traffic and maximum user load

Security Testing

- Security issues include: sensitive data that may remain on handsets, or passwords that are displayed on the screen
- Testing with automation is not feasible, what is required to be tested is the behavior of the actual handset and security designs vary among handsets – each device must be individually tested

Mobile Testing Techniques

Power Consumption/ Battery Life Testing

- Testing uncovers defects related to battery drainage caused by the application (device settings can also drain battery life and may make it difficult to determine the cause of the drain), the following are examples of different methods to test power consumption:
 - iPhone, iPod & iPad settings are adjusted; screen brightness, minimize use of location services, turn off push notifications, turn off other downloaded applications, fetch new data less frequently and turn off push mail; run the application to determine the rate it took for the battery life to decrease (testing executed manually without any testing tools)
 - Nokia Energy Profiler is a stand-alone test and measurement application which lets you monitor the battery consumption on target device

Defect Management and Tracking

Introduction

- Defect is the variance from a desired product attribute (it can be a wrong, missing or extra data).
- It can be of two types –
 - Defect from the **product or a variance from customer/user expectations.**
 - It is a flaw in the software system and has **no impact until it affects the user/customer and operational system.**
- With the knowledge of testing so far gained, you can now be able to categorize the defects you have found.
- Defects can be categorized into different types basing on the core issues they address.
- Some defects address security or database issues while others may refer to functionality or UI issues.

Types of Defect

- **Data Quality/Database Defects:** Deals with improper handling of data in the database.

- Examples:

- Values not deleted/inserted into the database properly
- Improper/wrong/null values inserted in place of the actual values

- **Critical Functionality Defects:** The occurrence of these bugs hampers the crucial functionality of the application. Examples: - Exceptions

- **Functionality Defects:** These defects affect the functionality of the application.

- Examples:

- All JavaScript errors
- Buttons like Save, Delete, Cancel not performing their intended functions
- A missing functionality (or) a feature not functioning the way it is intended to
- Continuous execution of loops

Types of Defect

Security Defects: Application security defects generally involve improper handling of data sent from the user to the application. These defects are the most severe and given highest priority for a fix.

Examples:

- Authentication: Accepting an invalid username/password
- Authorization: Accessibility to pages though permission not given

User Interface Defects: As the name suggests, the bugs deal with problems related to UI are usually considered less severe.

Examples:

- Improper error/warning/UI messages
- Spelling mistakes
- Alignment problems

Bug(Defect) Life Cycle

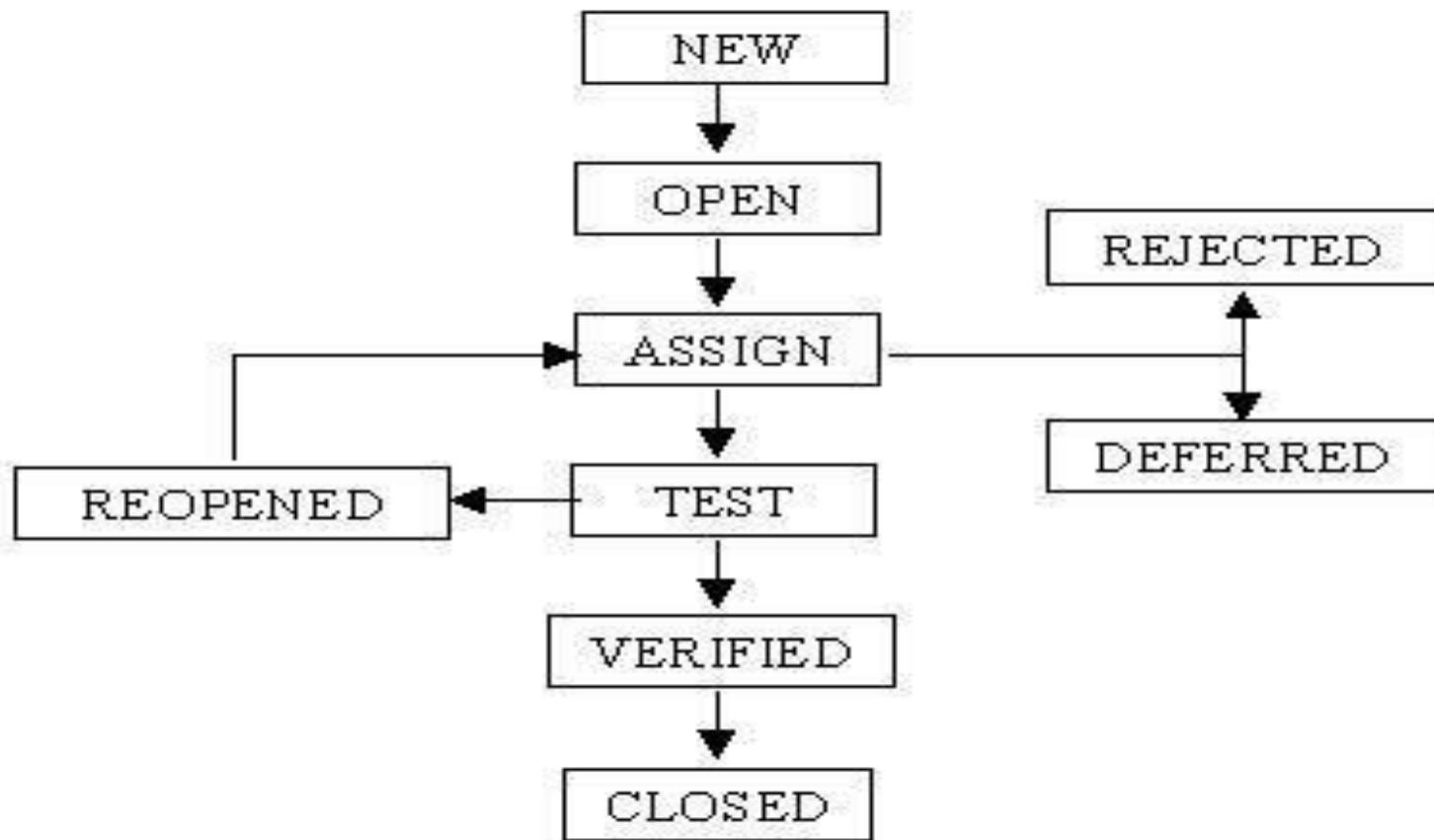
“A computer bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working correctly or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program’s source code or its design.”

The duration or time span between the first time defects is found and the time that it is closed successfully, rejected, postponed or deferred is called as ‘Defect Life Cycle’.

When a bug is discovered, it goes through several states and eventually reaches one of the terminal states, where it becomes inactive and closed.

The process by which the defect moves through the life cycle is depicted next slide.

Bug(Defect) Life Cycle



Bug(Defect) Life Cycle

- As you can see from above diagram, a defect's state can be divided into Open or Closed.

- When a bug reaches one of the Closed or Terminal states, its lifecycle ends. Each state has one or more valid states to move to.

- This is to ensure that all necessary steps are taken to resolve or investigate that defect. For example, a bug should not move from Submitted state to resolved state without having it open.

- In a typical scenario, as soon as a bug is identified, it is logged into the bug tracking system with status as Submitted. After ascertaining the validity of the defect, it is given the “Open” Status.

Defect Stages

- **New** : The Bug is newly found out and entered in the Bug tracking or Bug Reporting tool.
- **Open**: The Development or Test Lead reviews the defect. If it is determined to be a true defect, he or she adjusts the severity and priority and changes the status to open. A status of Open indicates that the defect is a true defect but that it has not yet been assigned to a developer for correction.
- **Assigned** : The bug is assigned to the Developer.
- **Tested** : The bug is tested by the Software tester.
- **Verified** : The bug is verified by the QA Lead.
- **Closed**: The tester verifies that the defect has been resolved and changes the status to Closed. A status of Closed indicates that the defect has been fixed and re-tested to the satisfaction of the person who first logged the defect.

Defect Stages(Cont...)

- Rejected:** If the Test Lead finds that the system is working according to the specifications or the defect is invalid as per the explanation from the development team, he/she rejects the defect and marks its status as 'Rejected'. A status of Rejected indicates that the defect is invalid, and therefore closed. No further work will be done on it.
- Deferred:** In some cases the client may determine that a particular defect stands less important and can be deferred to a later stage. In that case it may be marked with 'Deferred', with a comment indicating when it should be reviewed again. A status of Deferred indicates that no further work will be done on the defect until that later date.
- Reopened:** If after retesting the defect, the problem is not solved, the tester reopens changes the status to 'Reopened'. A status of re-opened indicates that the developer failed to satisfactorily fix the defect and that it needs to be re-assigned to a developer for fixing.

Defect Management Scenario

The typical defect lifecycle

- Submitted > Open > Assigned > Resolved > Migrated > Retest(Solved Defect) > Closed

Developer unable to fix defect in first attempt

- Submitted > Open > Assigned > Resolved > Migrated > Retest(Still Defect) > Reopen > Assigned > Resolved > Migrated > Retest(Solved Defect) > Closed

Defect is determined by test lead to be invalid

- Submitted > Rejected

Defect is determined by developer to be invalid

- Submitted > Open > Assigned > Rejected(Not a Defect)

Developer unable to recreate defect

- Submitted > Open > Assigned > Rejected(Not a Defect)

Client decides to defer correction of a defect

- Submitted > Open > Deferred(By Project Leader)

Defect Reporting

Defect reports are among the most important deliverables to come out of test. They are as important as the test plan and will have more impact on the quality of the product than most other deliverables from the test. Effective defect reports will:

- Reduce the number of defects returned from development
- Improve the speed of getting defect fixes
- Improve the credibility of test
- Enhance teamwork between test and development

IEEE 1044 BUG PROCESS

In IEEE standards 1044, and 1044.1, a generic process for reporting, managing and resolving bugs is laid out. The process includes four major steps

- Recognition
- Investigation
- Action
- Disposition

Within each step three activities take place

- Identifying
- Classifying
- Recording

Defect Report Fields

- You should provide enough detail while reporting the bug keeping in mind the people who will use it – test lead, developer, project manager, other testers, new testers assigned etc.

- This means that the report you will write should be concise, straight and clear.

- Following are the details your report should contain:

- Bug Title
- Bug identifier (number, ID, etc.)
- The application name or identifier and version / Test Case Id or Description
- The function, module, feature, object, screen, etc. where the bug occurred
- Environment (OS, Browser and its version)
- Bug Type or Category/Severity/Priority
- Bug status (Open, Pending, Fixed, Closed, Re-Open)
- Test case name/number/identifier
- Bug description
- Steps to Reproduce
- Actual Result
- Tester Comments

Defect Report Fields

- Bug Category: Security, Database, Functionality (Critical/General), UI
- Bug Severity: Severity with which the bug affects the application – Very High, High, Medium, Low, Very Low
- Bug Priority: Recommended priority to be given for a fix of this bug – Po, P1, P2, P3, P4, P5 (Po-Highest, P5-Lowest)

WHAT DOES THE TESTER DO WHEN THE DEFECT IS FIXED?

- Once the reported defect is fixed, the tester needs to re-test to confirm the fix. This is usually done by executing the possible scenarios where the bug can occur. Once retesting is completed, the fix can be confirmed and the bug can be closed. This marks the end of the bug life cycle.

Defect Severity

Severity is absolute and Customer-Focused. It is the extent to which the defect can affect the software. In other words it defines the impact that a given defect has on the system.

- **For example:** If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by an user is rare but the impact of application crashing is severe. So the severity is high but priority is low.

Severity can be of following types:

- **Critical:** The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable and there is no acceptable alternative method to achieve the required results then the severity will be stated as critical.

Defect Severity(Cont...)

Severity can be of following types:

- **Major (High):** The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable but there exists an acceptable alternative method to achieve the required results then the severity will be stated as major.
- **Moderate (Medium):** The defect that does not result in the termination, but causes the system to produce incorrect, incomplete or inconsistent results then the severity will be stated as moderate.
- **Minor (Low):** The defect that does not result in the termination and does not damage the usability of the system and the desired results can be easily obtained by working around the defects then the severity is stated as minor.
- **Cosmetic:** The defect that is related to the enhancement of the system where the changes are related to the look and feel of the application then the severity is stated as cosmetic.

Defect Priority

Priority is Relative and Business-Focused. Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest. The priority status is set based on the customer requirements.

- **For example:** If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it.

Priority can be of following types:

- **Low:** The defect is an irritant which should be repaired, but repair can be deferred until after more serious defect has been fixed.
- **Medium:** The defect should be resolved in the normal course of development activities. It can wait until a new build or version is created.
- **High:** The defect must be resolved as soon as possible because the defect is affecting the application or the product severely. The system cannot be used until the repair has been done.
- **Critical:** Extremely urgent, resolve immediately

Scenario of Defect Priority - Severity

- **High Priority & High Severity:** An error which occurs on the basic functionality of the application and will not allow the user to use the system.
(Eg. A site maintaining the student details, on saving record if it, doesn't allow to save the record then this is high priority and high severity bug.)

- **High Priority & Low Severity:** The spelling mistakes that happens on the cover page or heading or title of an application.

- **High Severity & Low Priority:** An error which occurs on the functionality of the application (for which there is no workaround) and will not allow the user to use the system but on click of link which is rarely used by the end user.

- **Low Priority and Low Severity:** Any cosmetic or spelling issues which is within a paragraph or in the report (Not on cover page, heading, title).

Cost of Bug Fixing

- Costs are logarithmic; they increase in size tenfold as the time increases.
- A bug found and fixed during the early stages – requirements or product spec stage can be fixed by a brief interaction with the concerned and might cost next to nothing.
- During coding, a swiftly spotted mistake may take only very less effort to fix.
- During integration testing, it costs the paperwork of a bug report and a formally documented fix, as well as the delay and expense of re-test.
- During system testing it costs even more time and may delay delivery.
- Finally, during operations it may cause anything from a nuisance to a system failure, possibly with catastrophic consequences in a safety critical system such as an aircraft or an emergency service.

Module - 4

Defect Tracking Tools

Or

Bug Tracking

Bug Tracking Tools

1. Bugzilla:



2. JIRA:



5. Redmine:



3. Mantis:



4. Trac:





Bug Tracking Tools

6. HP ALM/Quality Center:



7. FogBugz:



10. Zoho bug tracker:



8. IBM Rational ClearQuest:



9. Lighthouse:

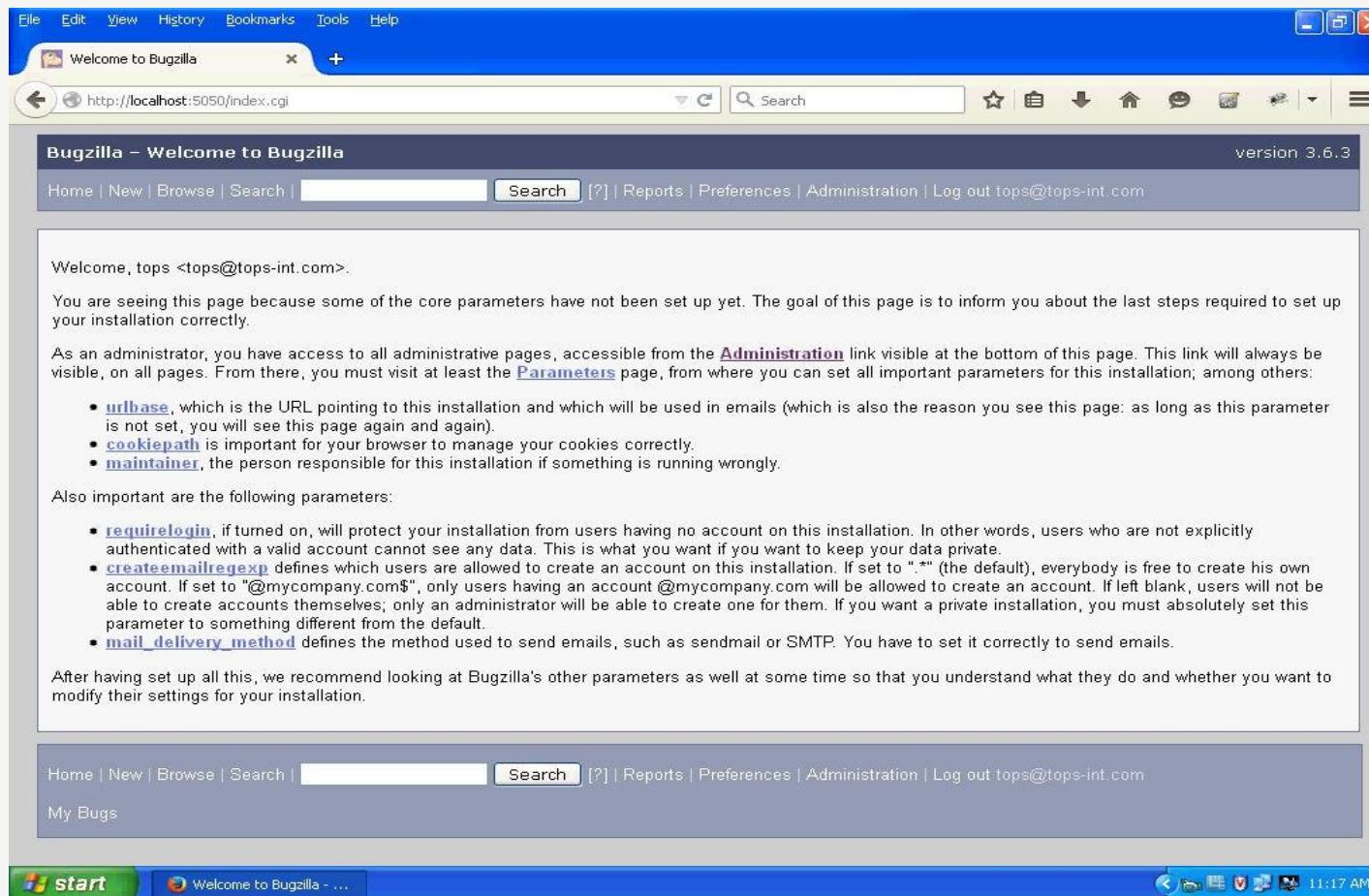


Bugzilla

- Bugzilla is an open-source issue/bug tracking system that allows developers effectively to keep track of outstanding problems with their product. It is written in Perl and uses MYSQL database.
- Bugzilla is a defect tracking tool, however it can be used as a test management tool as such it can be easily linked with other test case management tools like Quality Center, Testlink etc.
- This open bug-tracker enables users to stay connected with their clients or employees, to communicate about problems effectively throughout the data-management chain.
- Key features of Bugzilla includes
 - Advanced search capabilities
 - E-mail Notifications
 - Modify/file Bugs by e-mail
 - Time tracking
 - Strong security
 - Customization
 - Localization

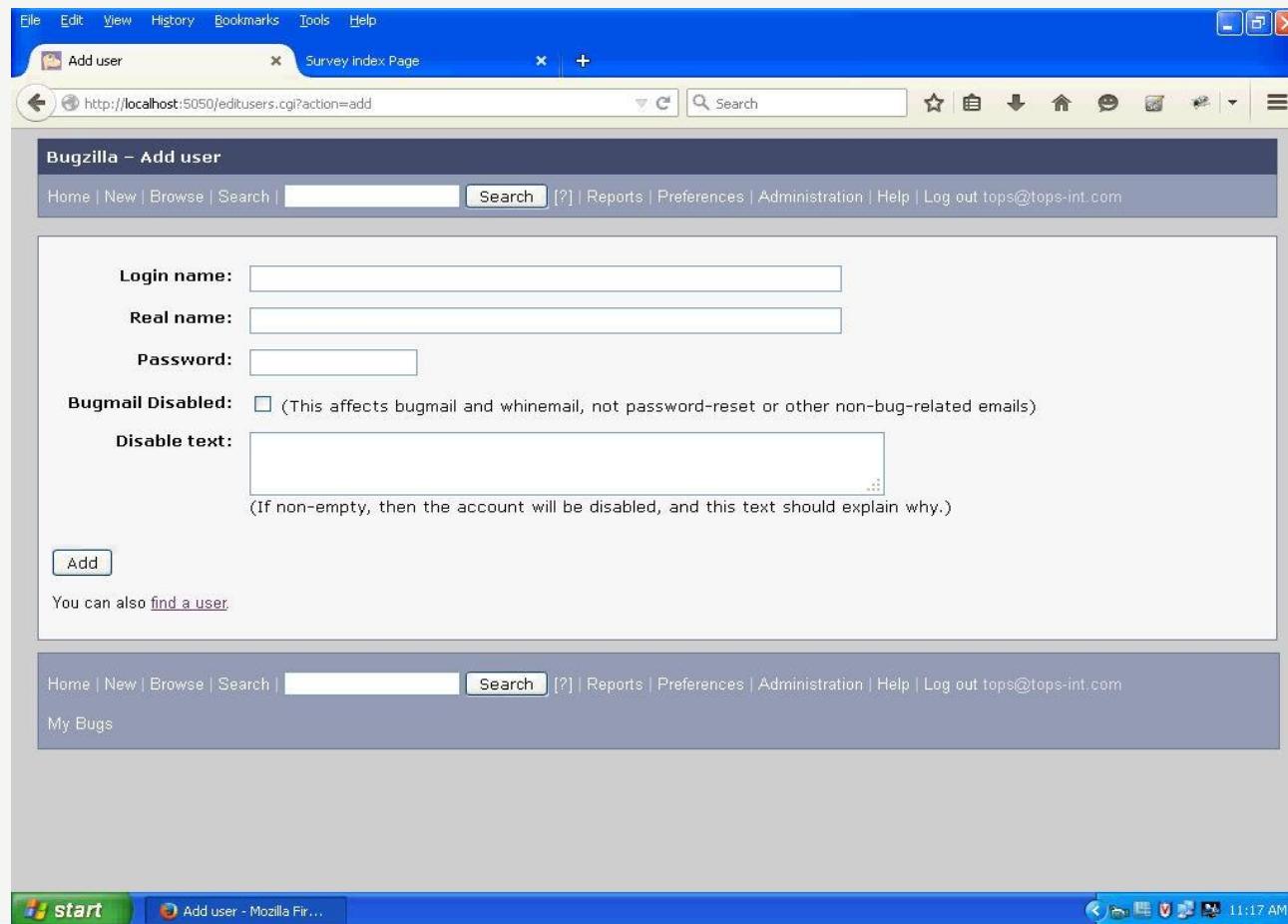
Bugzilla : Welcome

Screen



The screenshot shows a Microsoft Internet Explorer window displaying the Bugzilla Welcome screen. The title bar reads "Welcome to Bugzilla". The address bar shows the URL "http://localhost:5050/index.cgi". The page header says "Bugzilla – Welcome to Bugzilla" and "version 3.6.3". The main content area starts with a welcome message for user "tops <tops@tops-int.com>". It explains that some core parameters have not been set up yet and provides instructions for administrators to set them up. It lists several important parameters like `urlbase`, `cookiepath`, and `maintainer`. It also mentions other parameters like `requirelogin`, `createemailregexp`, and `mail_delivery_method`. At the bottom, it suggests looking at other parameters. The footer includes links for Home, New, Browse, Search, and Administration, along with a log out link for "tops@tops-int.com". The taskbar at the bottom shows the start button and the title "Welcome to Bugzilla - ...".

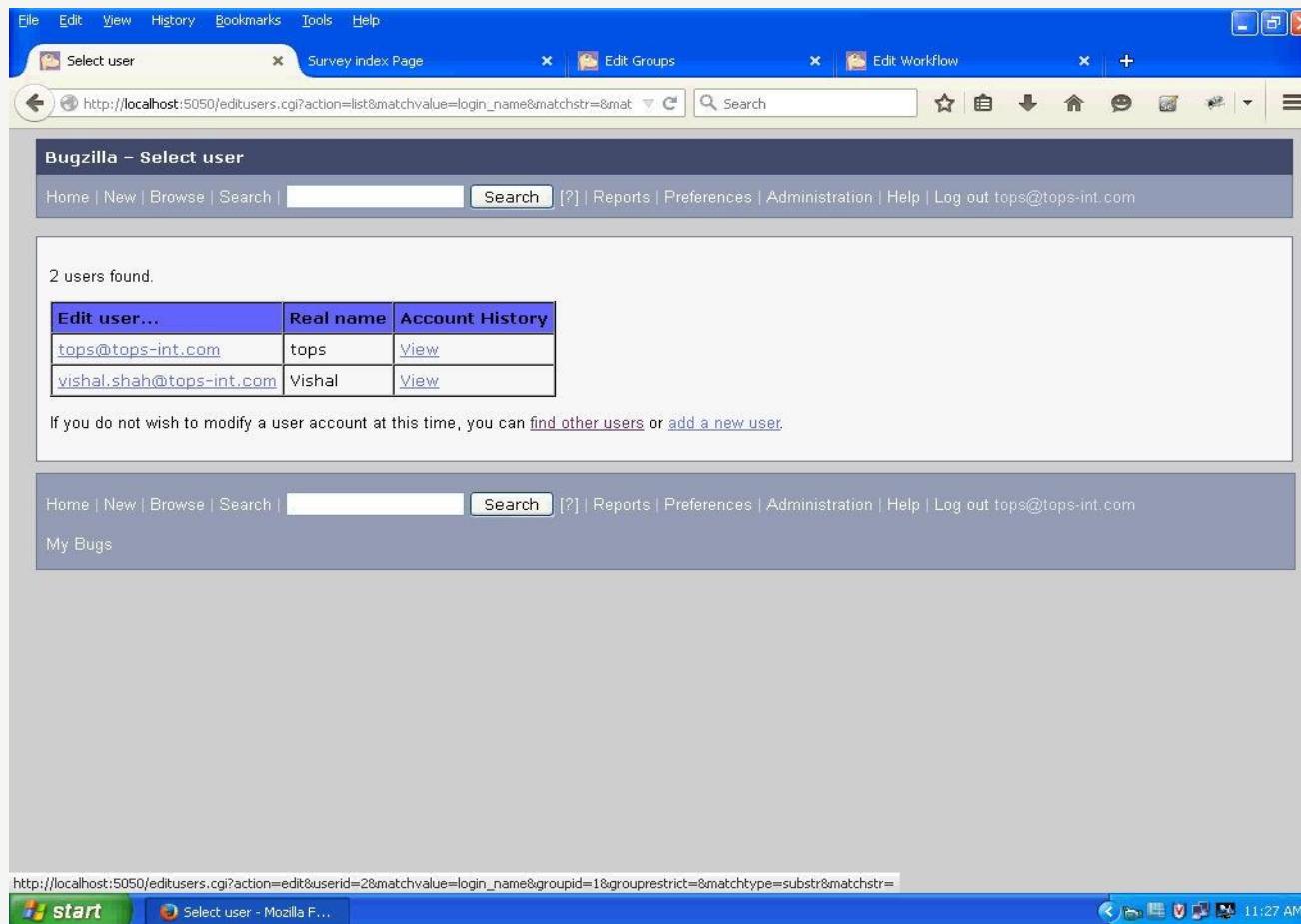
Bugzilla : Add User



The screenshot shows a Mozilla Firefox browser window with the following details:

- Title Bar:** File Edit View History Bookmarks Tools Help
Add user Survey index Page
- Address Bar:** http://localhost:5050/editusers.cgi?action=add
- Content Area:**
 - Bugzilla – Add user**
 - Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
 - Login name:**
 - Real name:**
 - Password:**
 - Bugmail Disabled:** (This affects bugmail and whinemail, not password-reset or other non-bug-related emails)
 - Disable text:**
(If non-empty, then the account will be disabled, and this text should explain why.)
 - Add** button
 - You can also [find a user](#).
- Bottom Navigation Bar:** Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
- Bottom Status Bar:** My Bugs
- Taskbar:** start | Add user - Mozilla Fir... | 11:17 AM

Bugzilla : Search User



The screenshot shows a Mozilla Firefox browser window with four tabs open:

- Select user
- Survey index Page
- Edit Groups
- Edit Workflow

The active tab is "Select user". The address bar shows the URL: `http://localhost:5050/editusers.cgi?action=list&matchvalue=login_name&matchstr=&mat`. The page title is "Bugzilla - Select user".

The main content area displays a table of users found:

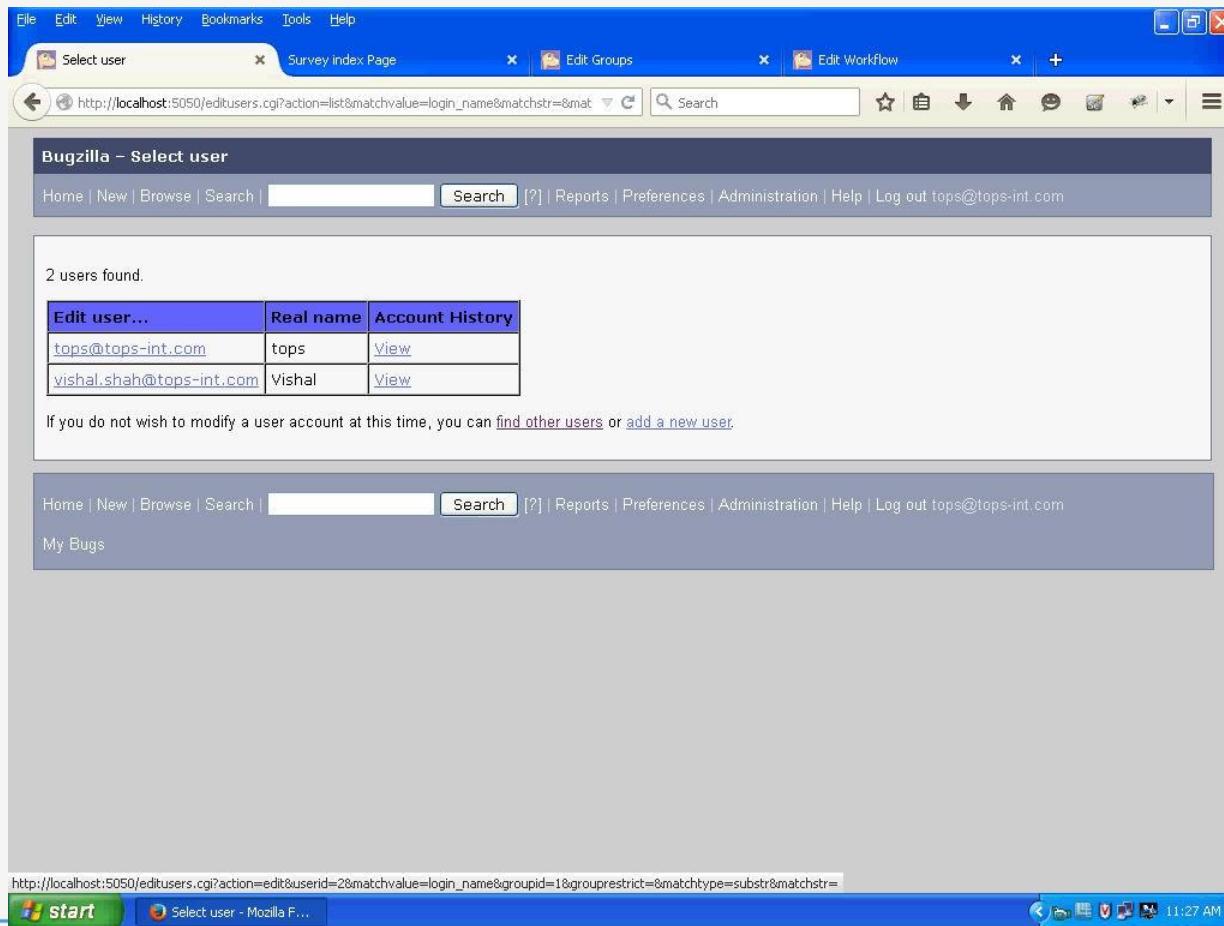
Edit user...	Real name	Account History
tops@tops-int.com	tops	View
vishal.shah@tops-int.com	Vishal	View

A message below the table states: "If you do not wish to modify a user account at this time, you can [find other users](#) or [add a new user](#)".

The bottom of the page includes a navigation bar with links: Home | New | Browse | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com

The status bar at the bottom of the browser window shows the full URL: `http://localhost:5050/editusers.cgi?action=edit&userid=2&matchvalue=login_name&groupid=1&grouprestrict=0&matchtype=substr&matchstr=`.

Select User



The screenshot shows a Mozilla Firefox browser window with three tabs open:

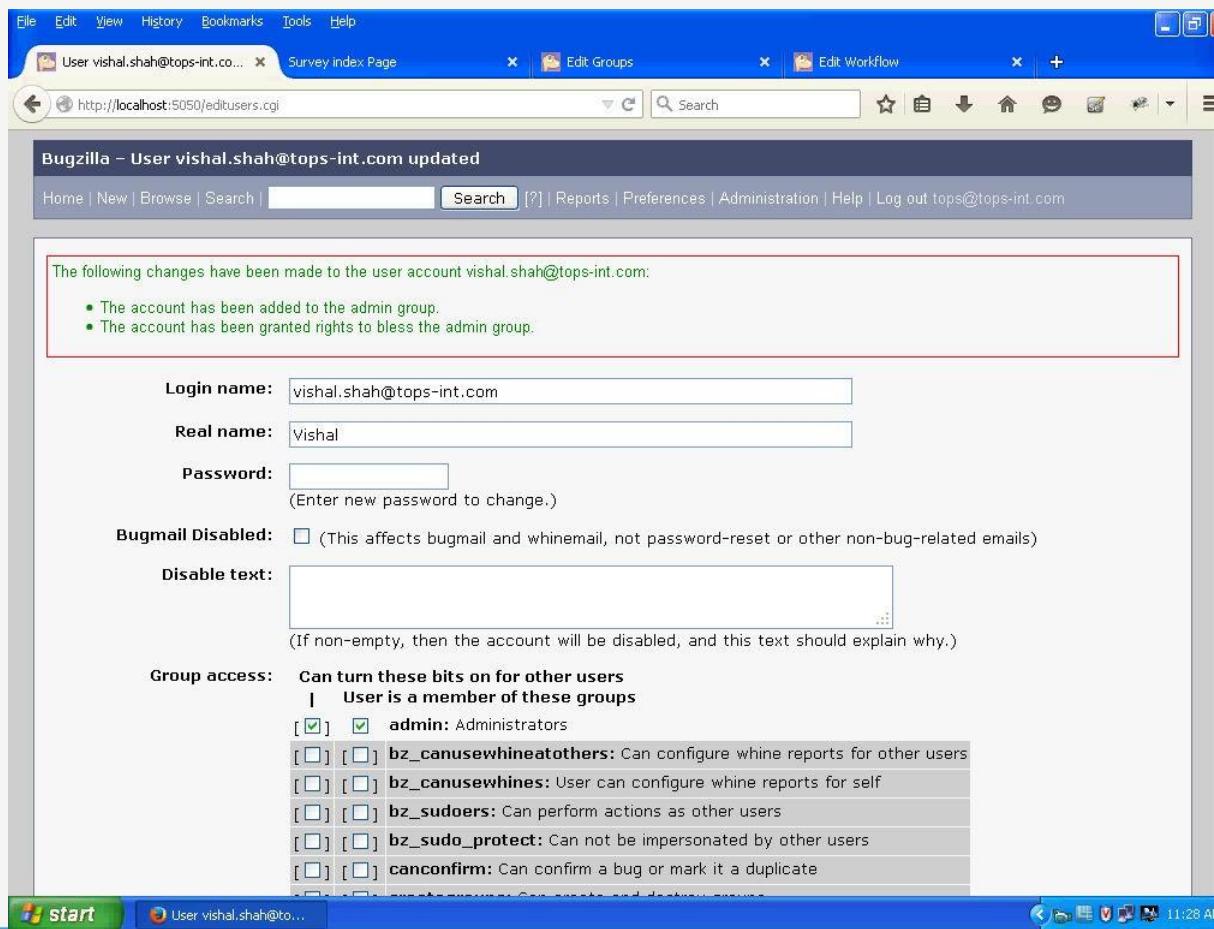
- Select user**: The active tab, displaying the Bugzilla "Select user" page. The URL is http://localhost:5050/editusers.cgi?action=list&matchvalue=login_name&matchstr=&mat. The page shows a table with two users found:

Edit user...	Real name	Account History
tops@tops-int.com	tops	View
vishal.shah@tops-int.com	Vishal	View

If you do not wish to modify a user account at this time, you can [find other users](#) or [add a new user](#).
- Survey index Page**
- Edit Groups**

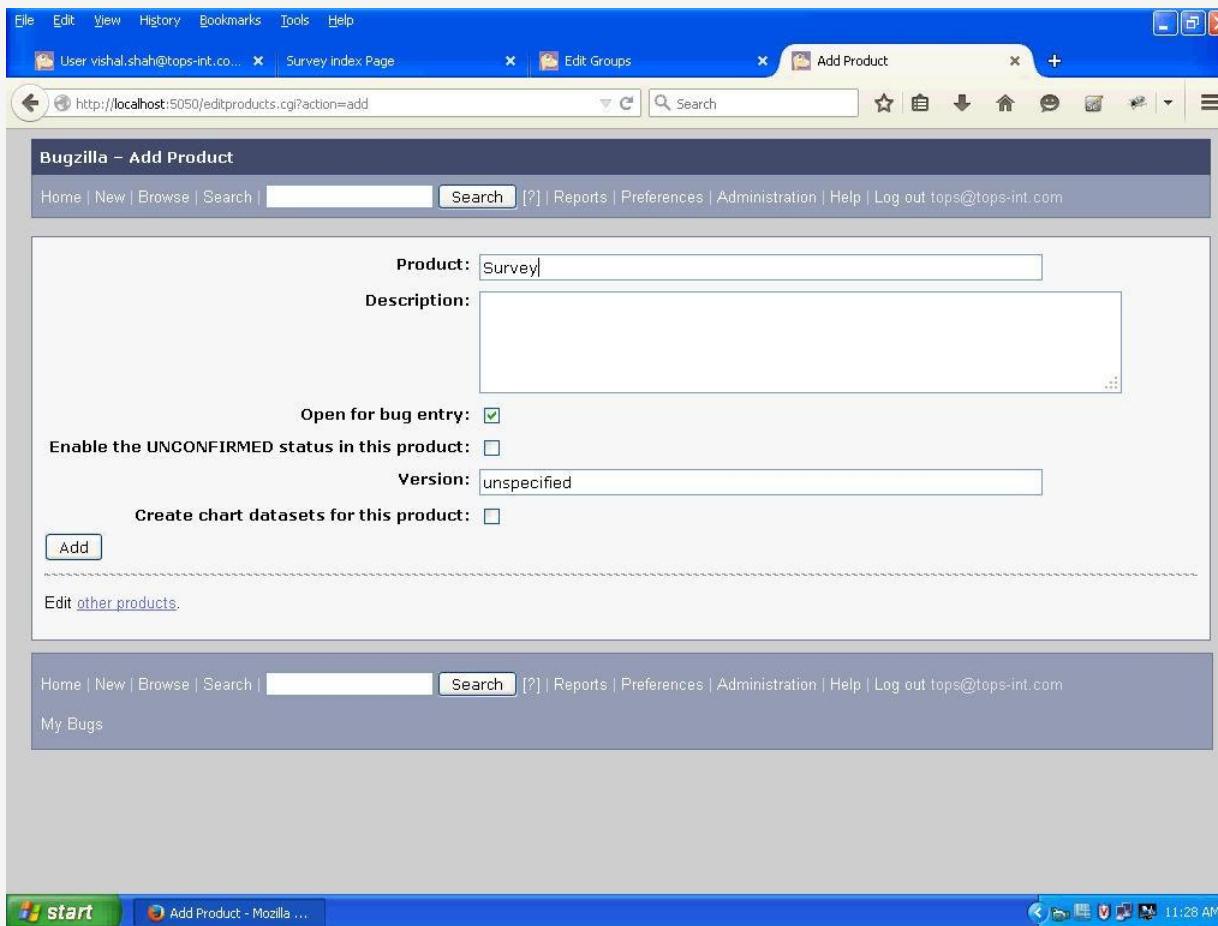
The browser's address bar also contains the URL http://localhost:5050/editusers.cgi?action=edit&userid=2&matchvalue=login_name&groupid=1&grouprestrict=&matchtype=substr&matchstr=.

Edit User



The screenshot shows a web browser window with the title "User vishal.shah@tops-int.co... Survey index Page Edit Groups Edit Workflow". The address bar displays "http://localhost:5050/editusers.cgi". The main content area is titled "Bugzilla – User vishal.shah@tops-int.com updated". It shows a message: "The following changes have been made to the user account vishal.shah@tops-int.com: • The account has been added to the admin group. • The account has been granted rights to bless the admin group." Below this, there are fields for "Login name" (vishal.shah@tops-int.com), "Real name" (Vishal), and "Password" (a placeholder field). A checkbox for "Bugmail Disabled" is checked. There is a "Disable text" input field with the note "(If non-empty, then the account will be disabled, and this text should explain why.)". Under "Group access", there is a section for "Can turn these bits on for other users" with a checkbox for "User is a member of these groups". Several checkboxes are listed, with "admin: Administrators" being checked. Other options include "bz_canusewhineatothers", "bz_canusewhines", "bz_sudoers", "bz_sudo_protect", and "canconfirm". The bottom of the window shows the Windows taskbar with icons for Start, File Explorer, Task View, and others.

Add product



File Edit View History Bookmarks Tools Help

User vishal.shah@tops-int.co... Survey index Page Edit Groups Add Product

http://localhost:5050/editproducts.cgi?action=add

Bugzilla – Add Product

Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com

Product: Survey

Description:

Open for bug entry:

Enable the UNCONFIRMED status in this product:

Version: unspecified

Create chart datasets for this product:

Add

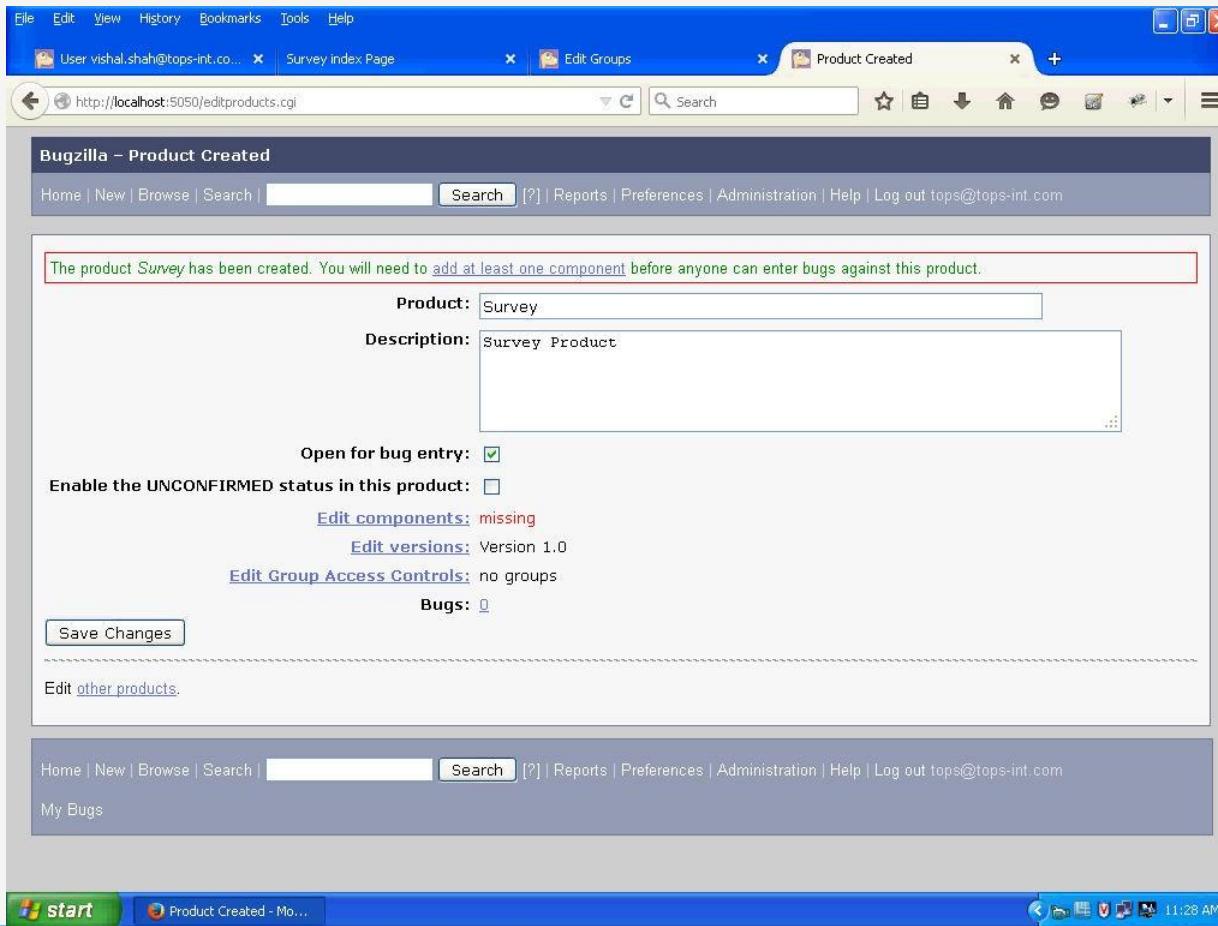
Edit other products.

Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com

My Bugs

start Add Product - Mozilla ... 11:28 AM

Product Added

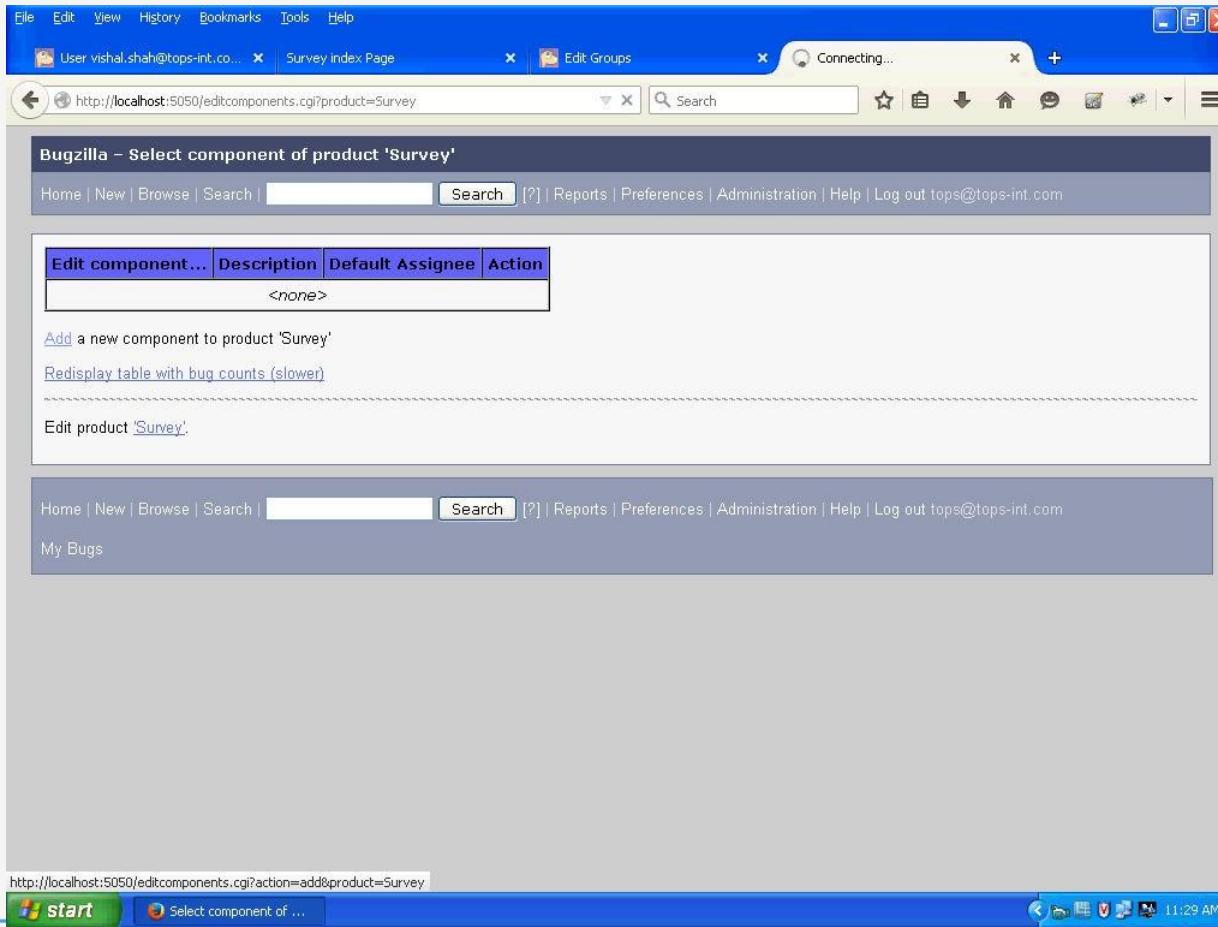


The screenshot shows a web browser window with three tabs: "User vishal.shah@tops-int.co...", "Survey index Page", and "Edit Groups". The active tab is "Product Created" at the URL <http://localhost:5050/editproducts.cgi>. The page title is "Bugzilla – Product Created". The main content area displays the following information:

Product: Survey
Description: Survey Product
Open for bug entry:
Enable the UNCONFIRMED status in this product:
[Edit components](#): missing
[Edit versions](#): Version 1.0
[Edit Group Access Controls](#): no groups
Bugs: 0
[Save Changes](#)

Below this, there is a link to "Edit other products". At the bottom of the page, there is a footer with links to "Home", "New", "Browse", "Search", "Reports", "Preferences", "Administration", "Help", and "Log out tops@tops-int.com". The status bar at the bottom of the browser window shows "Product Created - Mo..." and the time "11:28 AM".

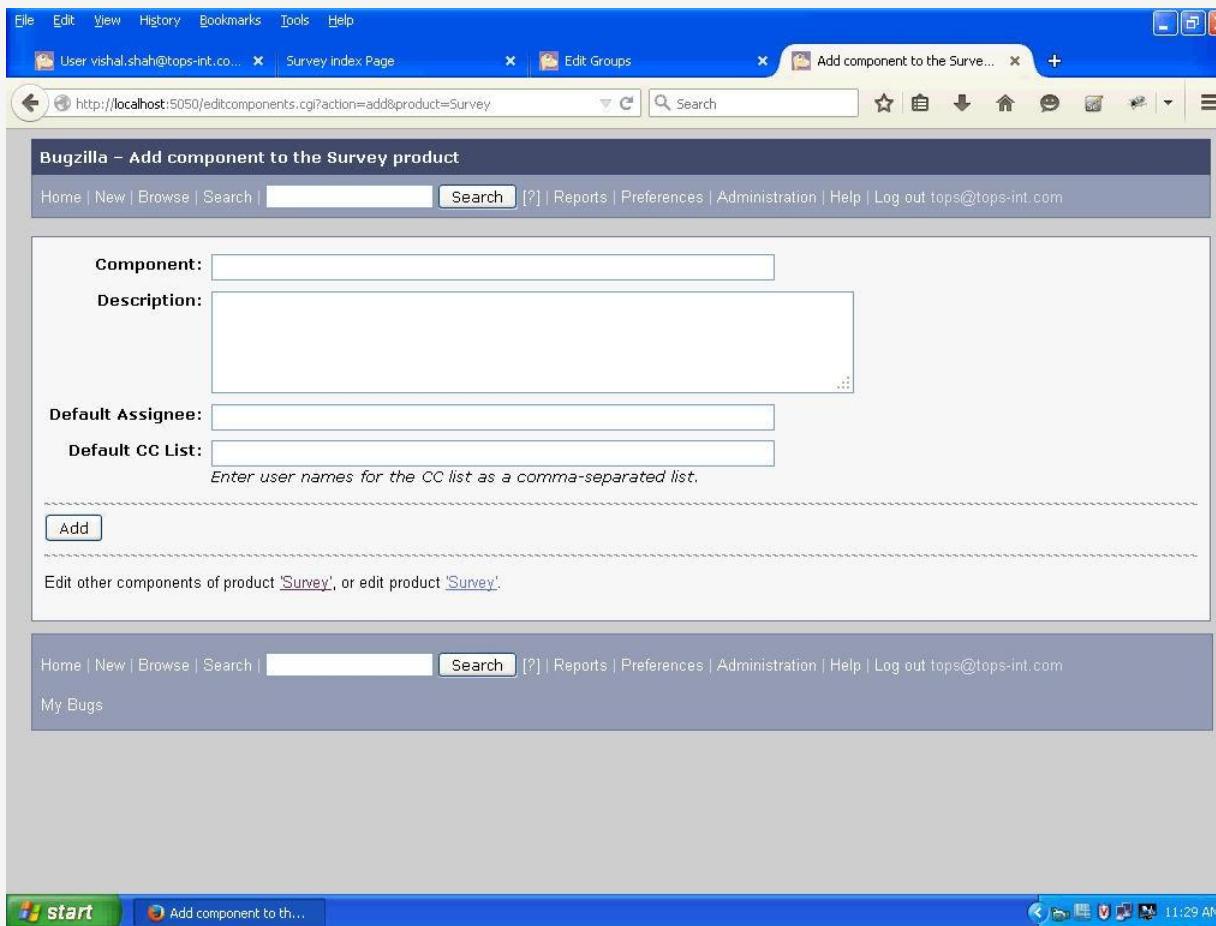
Select component on product



The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost:5050/editcomponents.cgi?product=Survey
- Title Bar:** Bugzilla – Select component of product 'Survey'
- Header:** Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
- Table:** A table with four columns: "Edit component...", "Description", "Default Assignee", and "Action". The "Edit component..." column is highlighted with a blue background. The "Action" column contains the text "<none>".
- Text Links:** Add a new component to product 'Survey', Redisplay table with bug counts (slower), Edit product 'Survey'.
- Footer:** Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
- Bottom Status Bar:** http://localhost:5050/editcomponents.cgi?action=add&product=Survey

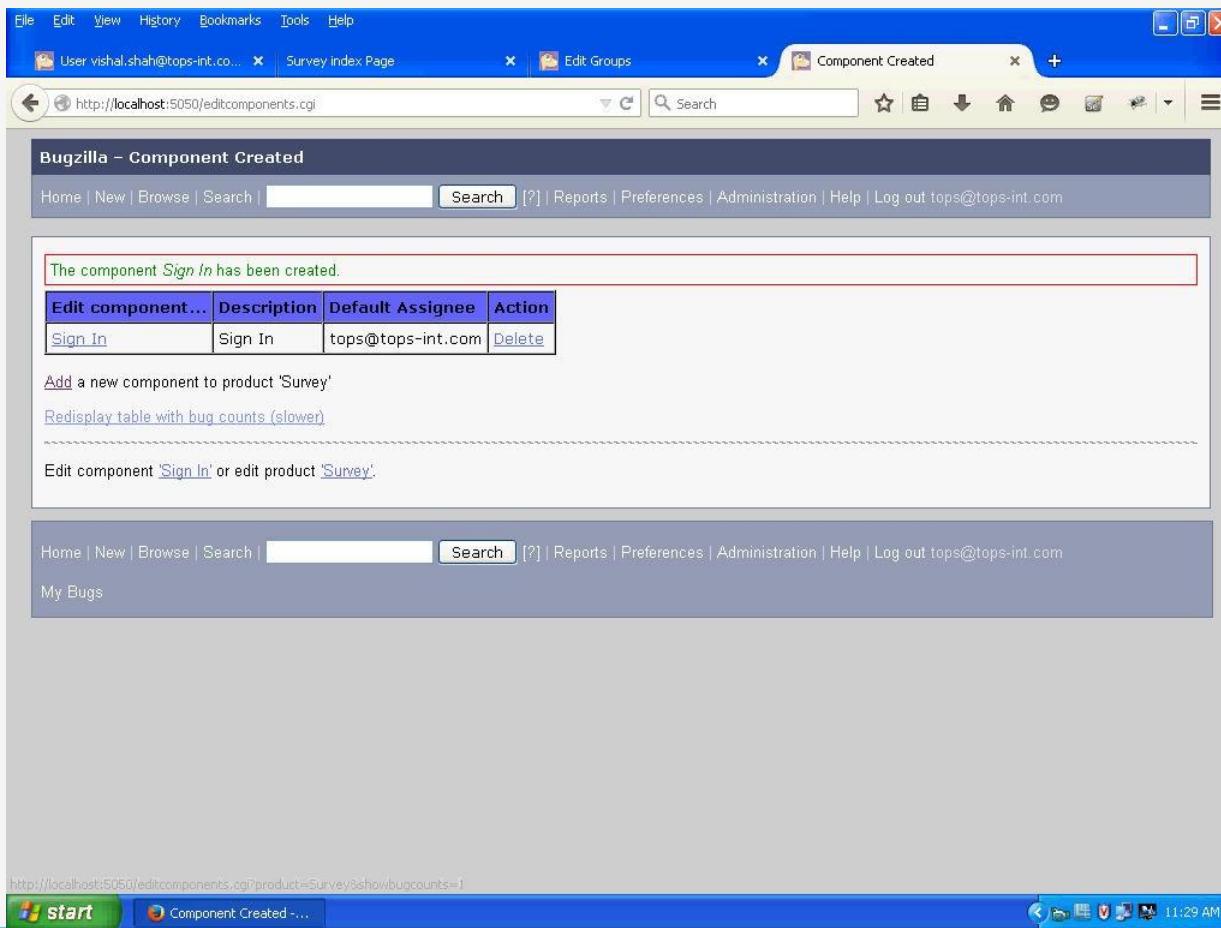
Add component to product



The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost:5050/editcomponents.cgi?action=add&product=Survey
- Title Bar:** Add component to the Survey product
- Content Area:**
 - Component:** [Text input field]
 - Description:** [Text area]
 - Default Assignee:** [Text input field]
 - Default CC List:** [Text input field]
Enter user names for the CC list as a comma-separated list.
 - Add:** [Submit button]
 - Link:** Edit other components of product 'Survey', or edit product 'Survey'.
- Bottom Navigation:** Home | New | Browse | Search | Search | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
- Bottom Status Bar:** My Bugs

Component Added

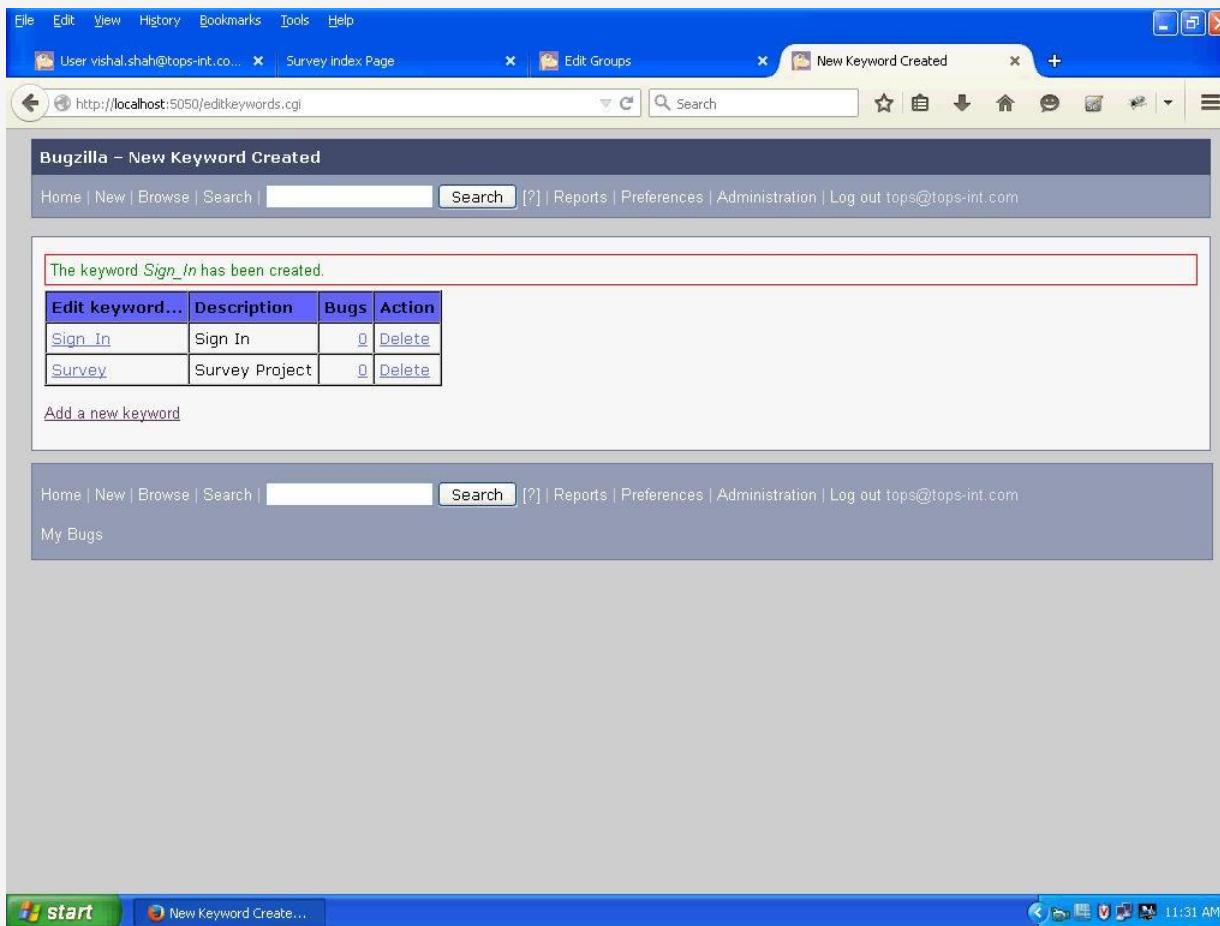


The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost:5050/editcomponents.cgi
- Title Bar:** Survey index Page, Edit Groups, Component Created
- Content Area:**
 - Bugzilla – Component Created**
 - Message:** The component *Sign In* has been created.
 - Table:** A table showing the newly created component.

Edit component...	Description	Default Assignee	Action
Sign In	Sign In	tops@tops-int.com	Delete
 - Links:** Add a new component to product 'Survey', Redisplay table with bug counts (slower), Edit component [Sign In](#) or edit product [Survey](#).
 - Footer:** Home | New | Browse | Search | Search | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
- Taskbar:** Shows the URL http://localhost:5050/editcomponents.cgi?product=Survey&show=bugcounts=1, the Windows Start button, and the title Component Created - ...

New Keyword Added

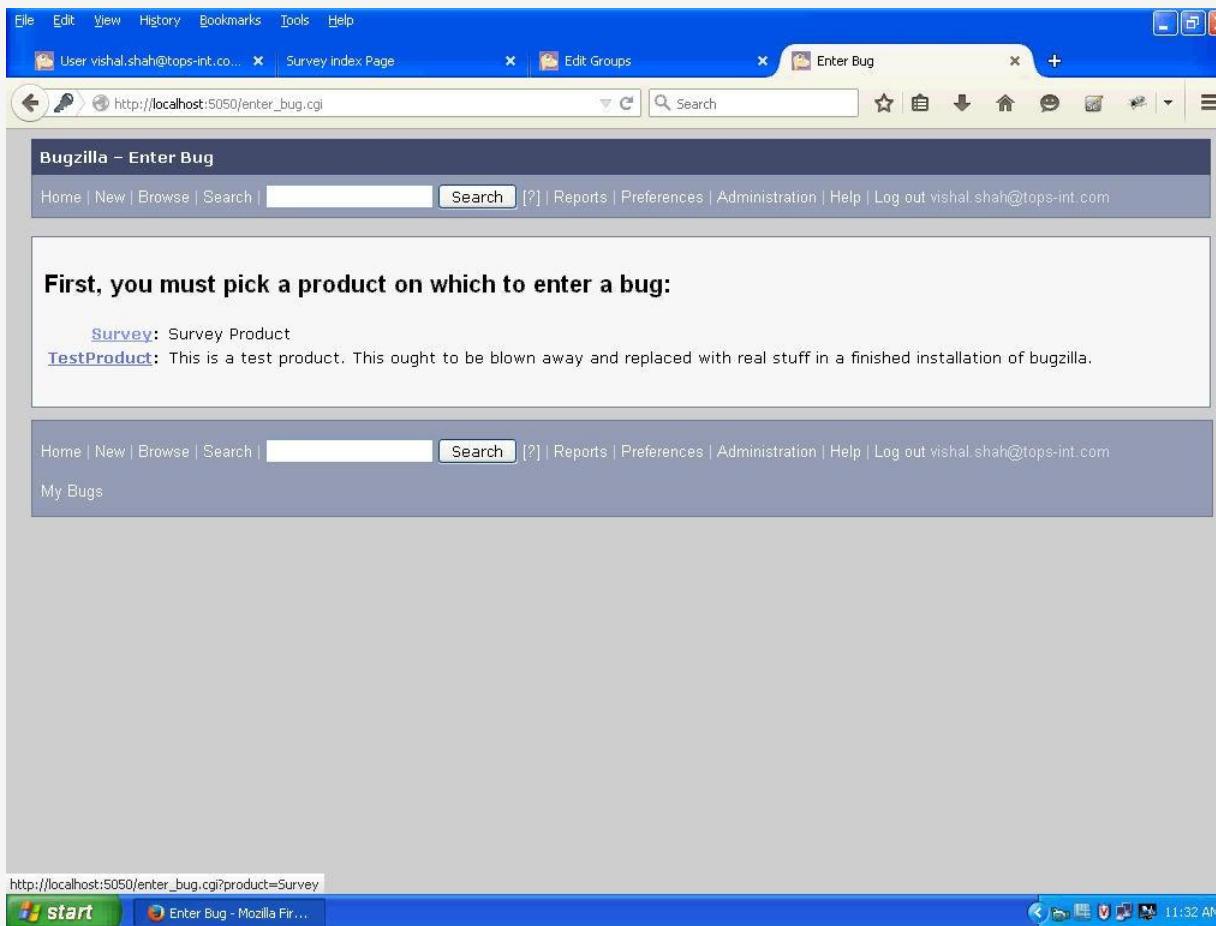


The screenshot shows a web browser window with the following details:

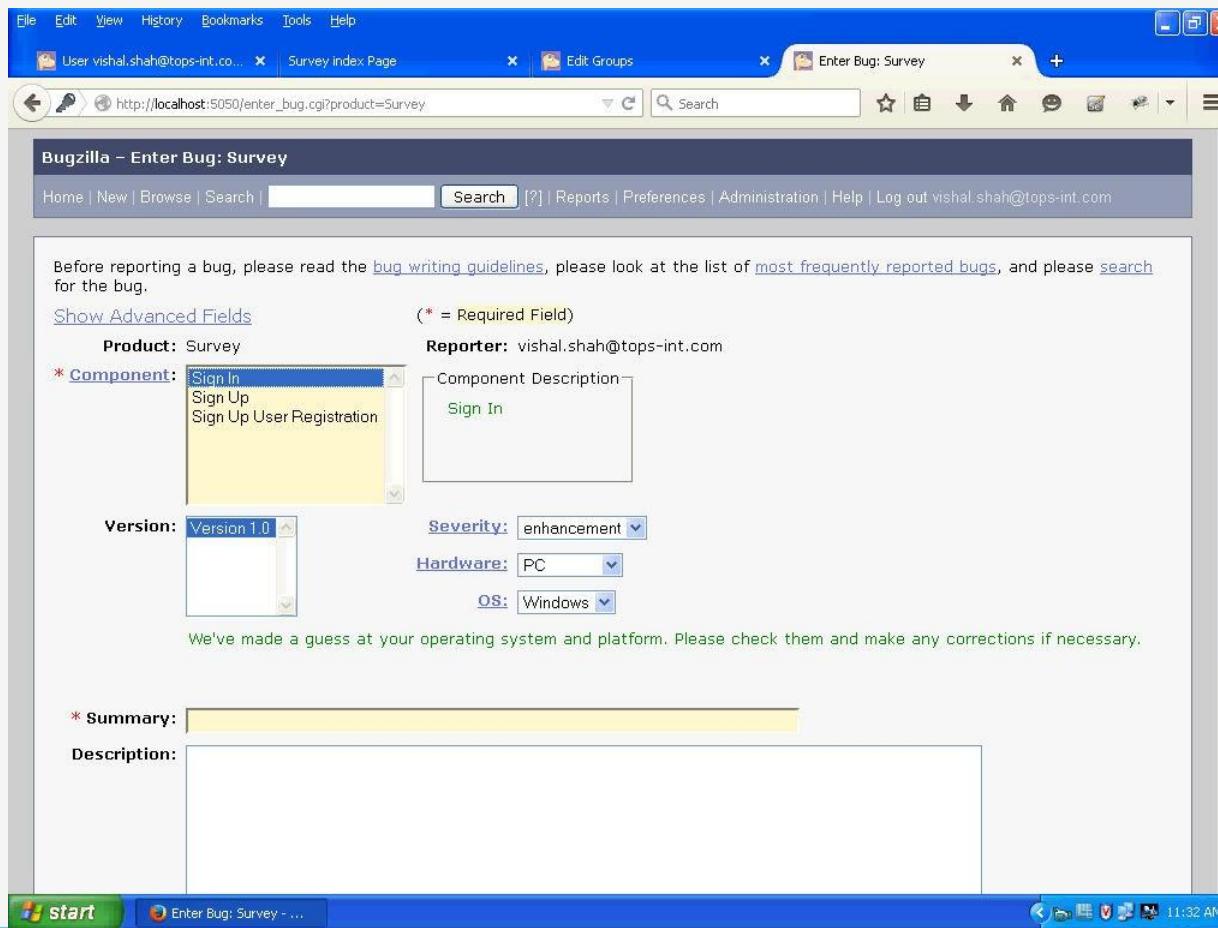
- Title Bar:** File Edit View History Bookmarks Tools Help
- Tab Bar:** User vishal.shah@tops-int.co... Survey index Page Edit Groups New Keyword Created
- Address Bar:** http://localhost:5050/editkeywords.cgi
- Content Area:**
 - Bugzilla – New Keyword Created**
 - Home | New | Browse | Search | Search | Reports | Preferences | Administration | Log out tops@tops-int.com
 - A message box: "The keyword *Sign_In* has been created."
 - A table:

Edit keyword...	Description	Bugs	Action
Sign_In	Sign In	0	Delete
Survey	Survey Project	0	Delete
 - [Add a new keyword](#)
- Bottom Navigation:** Home | New | Browse | Search | Search | Reports | Preferences | Administration | Log out tops@tops-int.com
- Bottom Status Bar:** start New Keyword Create... 11:31 AM

Select Product before enter new Bug



New Bug entry Main Page



File Edit View History Bookmarks Tools Help

User vishal.shah@tops-int.co... Survey index Page Edit Groups Enter Bug: Survey

http://localhost:5050/enter_bug.cgi?product=Survey

Bugzilla – Enter Bug: Survey

Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com

Before reporting a bug, please read the [bug writing guidelines](#), please look at the list of [most frequently reported bugs](#), and please [search](#) for the bug.

(* = Required Field)

Show Advanced Fields

Product: Survey **Reporter:** vishal.shah@tops-int.com

*** Component:** Sign In
Sign Up
Sign Up User Registration

Version: Version 1.0 **Severity:** enhancement

Hardware: PC **OS:** Windows

Component Description
Sign In

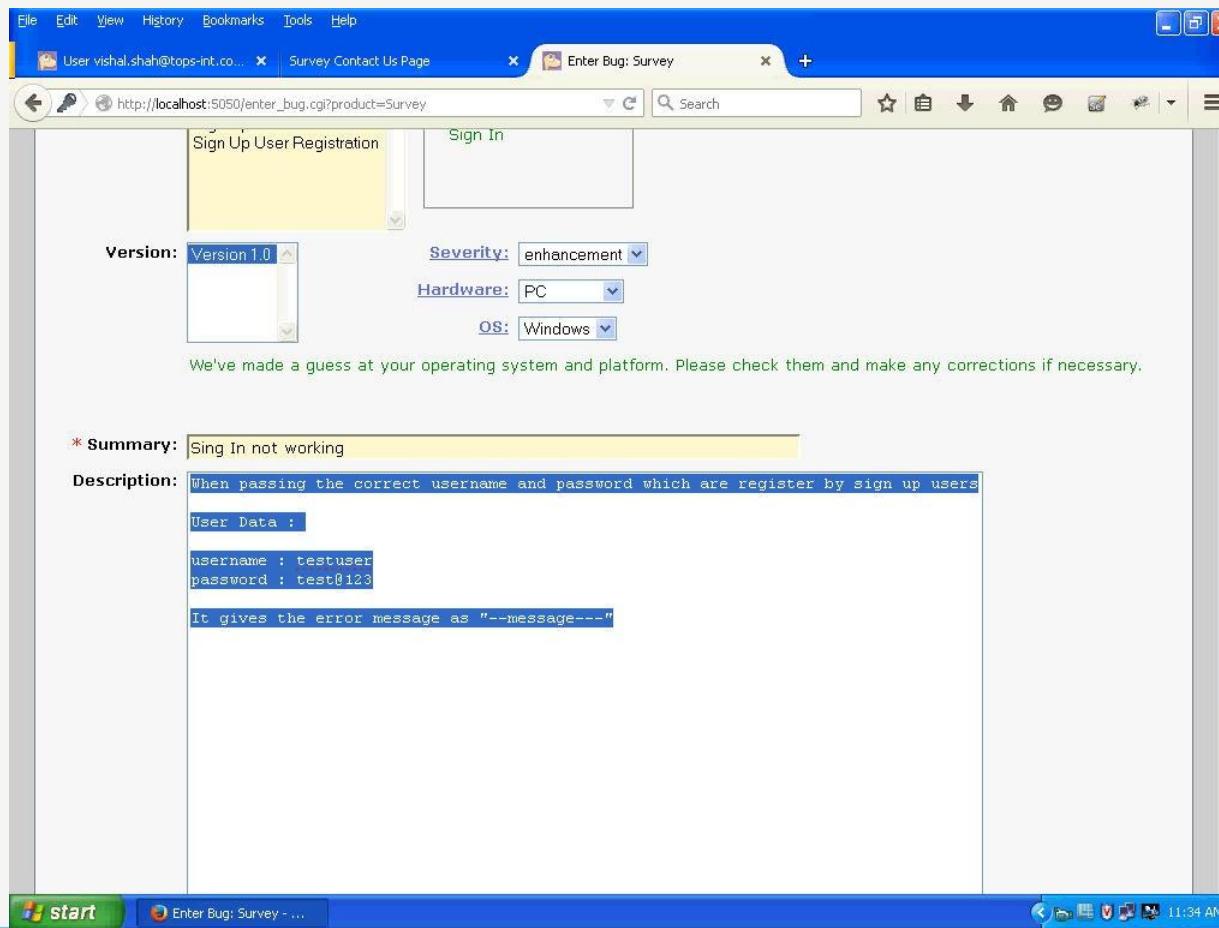
We've made a guess at your operating system and platform. Please check them and make any corrections if necessary.

*** Summary:**

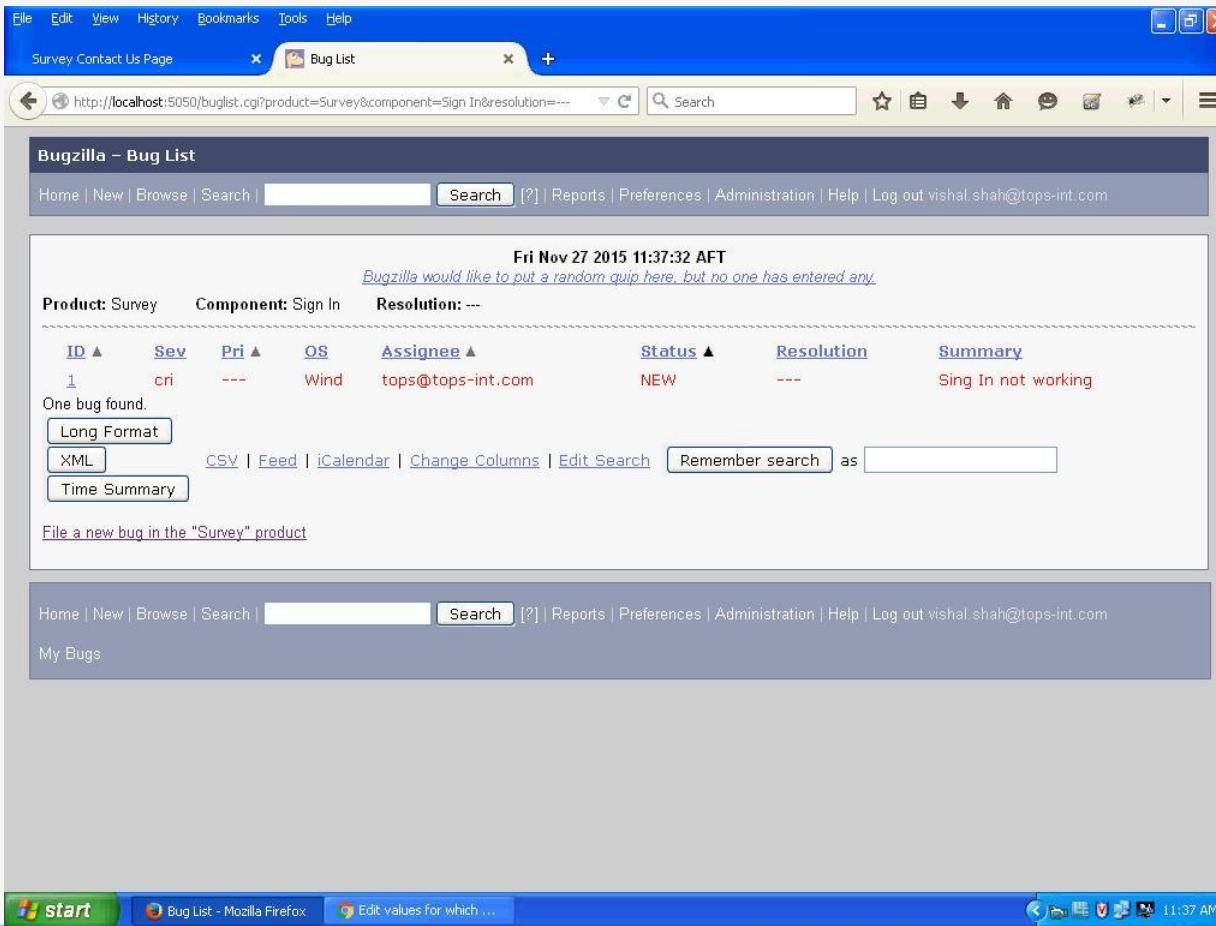
Description:

start Enter Bug: Survey - ... 11:32 AM

Add details into the new bug entry



Bugs which were entered



File Edit View History Bookmarks Tools Help

Survey Contact Us Page Bug List

http://localhost:5050/buglist.cgi?product=Survey&component=Sign+In&resolution=---

Search

Bugzilla – Bug List

Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com

Fri Nov 27 2015 11:37:32 AFT
Bugzilla would like to put a random quip here, but no one has entered any.

Product: Survey Component: Sign In Resolution: ---

ID ▲	Sev ▲	Pri ▲	OS	Assignee ▲	Status ▲	Resolution	Summary
1	cri	---	Wind	tops@tops-int.com	NEW	---	Sing In not working

One bug found.

Long Format CSV | Feed | iCalendar | Change Columns | Edit Search Remember search as

Time Summary

File a new bug in the "Survey" product

Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com

My Bugs

start Bug List - Mozilla Firefox Edit values for which ... 11:37 AM

Assign the Bug

File Edit View History Bookmarks Tools Help

Survey Contact Us Page x Bug 1 – Sing In not working +

http://localhost:5050/show_bug.cgi?id=1

Search

Bug 1 - Sing In not working ([edit](#))

Status: NEW ([edit](#))

Reported: 2015-11-27 11:36 AFT by [vishal](#)

Modified: 2015-11-27 11:36 AFT ([History](#))

CC List: Add me to CC list
0 users ([edit](#))

Save Changes

Product: Survey

Component: Sign In

Version: Version 1.0

Platform: All Windows

Importance: High critical

Assigned To: tops ([edit](#))

See Also: Add Bug URLs:

URL: http://survey-javatops.rhcloud.com/contact.jsp

Keywords:

Depends on:

Blocks:

Show dependency tree / graph

Orig. Est.	Current Est.	Hours Worked	Hours Left	%Complete	Gain	Deadline
0.0	0.0	0.0 + 0	0.0	0	0.0	(YYYY-MM-DD)

Summarize time (including time for bugs blocking this bug)

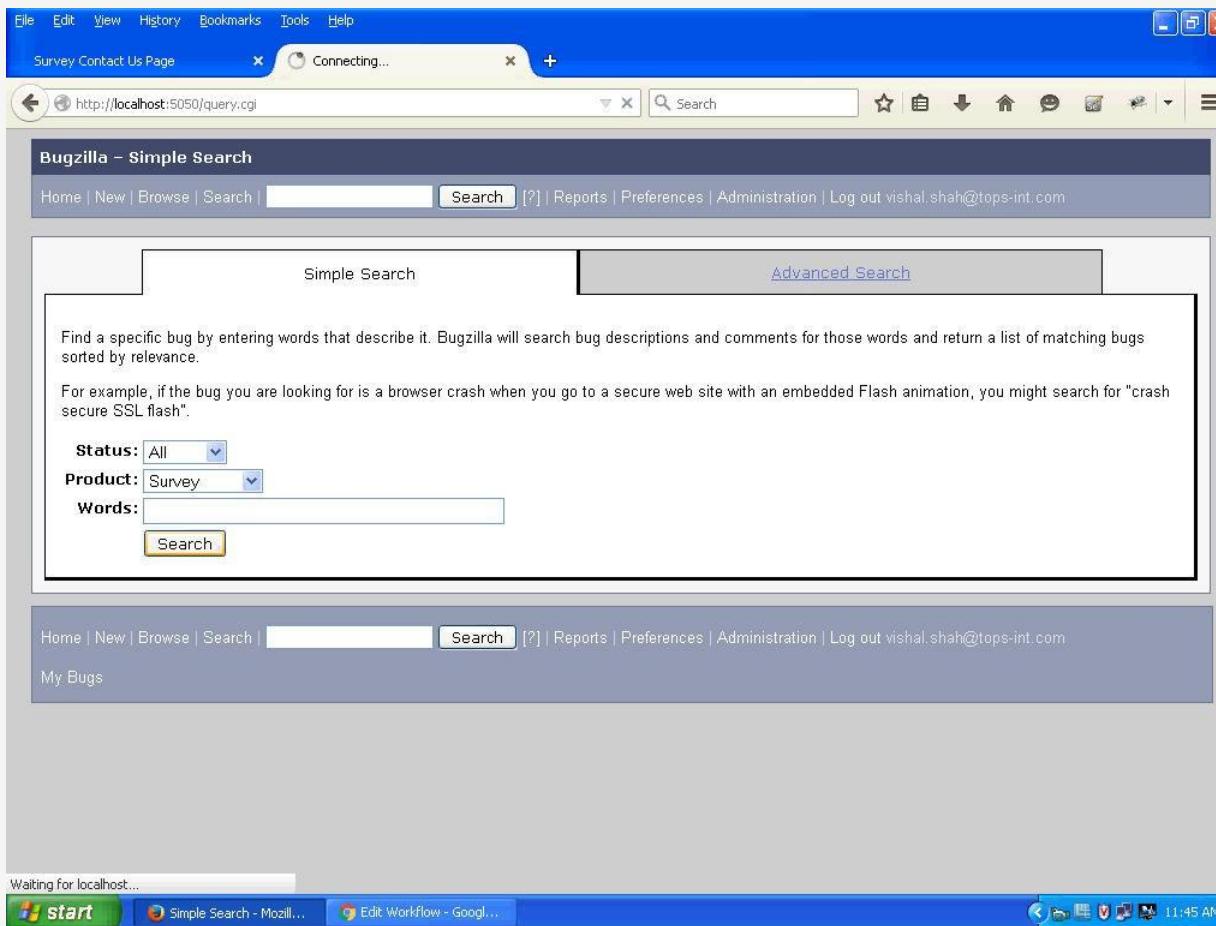
Attachments

Add an attachment (proposed patch, testcase, etc.)

Additional Comments:

start Bug 1 – Sing In not w... Edit values for which ... 11:38 AM

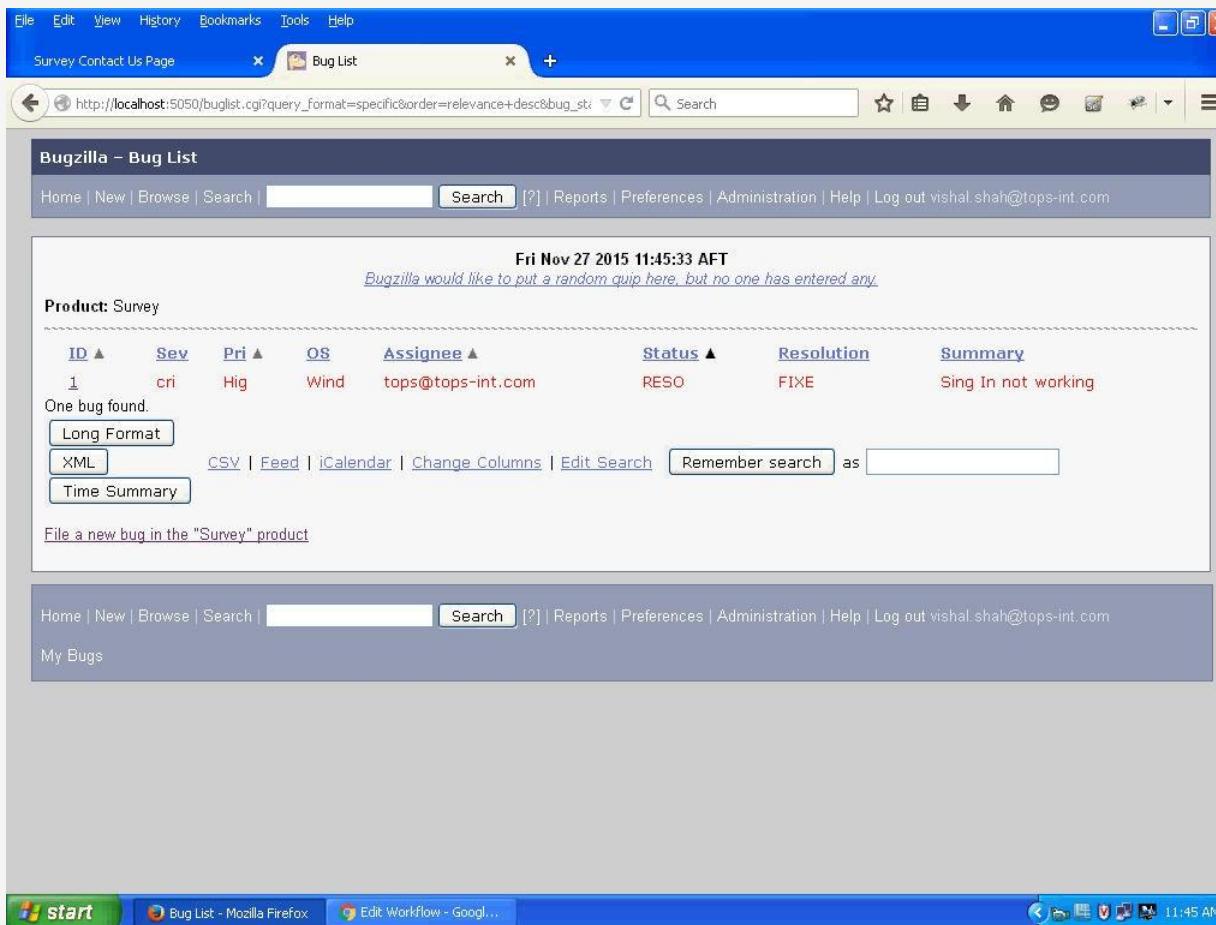
Search the Entered Bug



The screenshot shows a web browser window with the following details:

- Title Bar:** File Edit View History Bookmarks Tools Help
Survey Contact Us Page x Connecting... +
- Address Bar:** http://localhost:5050/query.cgi
- Page Title:** Bugzilla – Simple Search
- Header:** Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Log out vishal.shah@tops-int.com
- Content Area:**
 - Simple Search:** A section containing instructions and search filters.
 - Instructions: "Find a specific bug by entering words that describe it. Bugzilla will search bug descriptions and comments for those words and return a list of matching bugs sorted by relevance."
 - For example, if the bug you are looking for is a browser crash when you go to a secure web site with an embedded Flash animation, you might search for "crash secure SSL flash".
 - Filters:
 - Status: All
 - Product: Survey
 - Words:
 - Search button:
 - Advanced Search:** A link to the advanced search interface.
- Footer:** Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Log out vishal.shah@tops-int.com
- Bottom Status Bar:** My Bugs
- Bottom Footer:** Waiting for localhost...
- Taskbar:** start button, Simple Search - Mozilla..., Edit Workflow - Google..., 11:45 AM

Resolved Bug Status



Bugzilla – Bug List

Fri Nov 27 2015 11:45:33 AFT
Bugzilla would like to put a random quip here, but no one has entered any.

Product: Survey

ID ▲	Sev ▲	Pri ▲	OS	Assignee ▲	Status ▲	Resolution	Summary
1	cri	Hig	Wind	tops@tops-int.com	RESO	FIXE	Sing In not working

One bug found.

Long Format XML CSV Feed iCalendar Change Columns Edit Search Remember search as

Time Summary

[File a new bug in the "Survey" product](#)

Home | New | Browse | Search | [?] | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com

My Bugs

start Bug List - Mozilla Firefox Edit Workflow - Google... 11:45 AM

Module – 5 [Automation Testing Tools]

Agenda

- Fundamental of Automation

- Testing

- Automation Framework

- Load Runner Up

- Selenium IDE

Fundamental of Automation Testing

Introduction

- Test automation 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.
- Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.
- Although manual tests may find many defects in a software application, it is a laborious and time consuming process.
- In addition it may not be effective in finding certain classes of defects.
- Test automation is a process of writing a computer program to do testing that would otherwise need to be done manually.
- Once tests have been automated, they can be run quickly.
- This is often the most cost effective method for software products that have a long maintenance life, because even minor patches over the lifetime of the application can cause features to break which were working at an earlier point in time.

Approach to Test Automation

- **Code-driven testing:** The public (usually) interfaces to classes, modules, or libraries are tested with a variety of input arguments to validate that the results that are returned are correct.

- **Graphical user interface testing:** A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.

Factors of Automated Test Tools

- Examine your current testing process and determine where it needs to be adjusted for using automated test tools.
- Be prepared to make changes in the current ways you perform testing.
- Involve people who will be using the tool to help design the automated testing process.
- Create a set of evaluation criteria for functions that you will want to consider when using the automated test tool. These criteria may include the following:
 - Test repeatability
 - Criticality/risk of applications
 - Operational simplicity
 - Ease of automation
 - Level of documentation of the function (requirements, etc.)
- Examine your existing set of test cases and test scripts to see which ones are most applicable for test automation.
- Train people in basic test-planning skills.

Challenges of Test Automation

- Buying the Wrong Tool
- Inadequate Test Team Organization
- Lack of Management Support
- Incomplete Coverage of Test Types by the selected tool
- Inadequate Tool Training
- Difficulty using the tool
- Lack of a Basic Test Process or Understanding of What to Test
- Lack of Configuration Management Processes
- Lack of Tool Compatibility and Interoperability
- Lack of Tool Availability

Why Automation?

- **Speed** – Automation Scripts run very fast when compared to human users
- **Reliable** – Tests perform precisely the same operations each time they are run, thereby eliminating human error.
- **Repeatable** – We can test how the application reacts after repeated execution of the same operation
- **Comprehensive** – We can build a suite of tests that covers every feature in our application
- **Reusable** – We can reuse tests on different versions of an application, even if the user interface changes.
- **Automate your testing procedure when you have lot of regression work.**
- **Automate your load testing work for creating virtual users to check load capacity of your application.**
- **Automate your testing work when your GUI is almost frozen but you have lot of frequently functional changes.**

When to Automate?

Which Software Testing should be Automated?

- Test that need to be Execute of every build of the Application

- Test that use Multiple Data Value (Retesting & Regression Testing)

- Test that Required data From Application intimates(G.U.I. Attributes)

- Load and stress Testing

Which Software should not be Automate?

- Usability testing one time testing

- Quick look Tester as soon as possible testing Ad-hoc testing /Random testing

- Customer Requirement are frequently changing

When start the Automation Testing?

Manual tests are robust and well organized

- Strong, manual test script architecture will lend itself to creating a strong automation architecture. Ad hoc testing will lead to ad hoc automation tests, which may not be as useful as well planned tests.

Application GUI is fairly stable

- If you are starting with record/playback automation, the stability of the GUI (graphical user interface) becomes even more critical. If your development team is still designing the user interface (i.e., where different fields go, what they will be called, what types of controls will be used for each field) your automation tests are going to be playing catch up with the changes.

Application is structurally sound

- Every automated test failure requires some analysis to see if the application has a defect or if the test needs an update. The time required for these analyses can really add up especially when the overall framework of the application is not completely baked.

Risk in Automation Testing

- Manual Testing of **all work flows, all fields , all negative scenarios** is time and cost consuming
- It is **difficult to test for multi lingual sites manually**
- Automation does **not** require **Human intervention**. You can run automated test unattended (overnight)
- Automation **increases speed of test execution**
- Automation helps **increase Test Coverage**
- **Manual Testing** can become **boring** and hence **error prone**

Instead of relying 100% on either manual or automation use the best combination of manual and automation testing.

- This is the best solution (I think) for every project. Automation suite will not find all the bugs and cannot be a replacement for real testers. Ad-hoc testing is also necessary in many cases.

Which Test Cases to Automate?

Test cases to be automated can be selected using the following criterion to increase the automation ROI (Return on Investment)

- **High Risk - Business Critical test cases**
- Test cases that are **executed repeatedly**
- Test Cases that are very **tedious** or difficult to perform manually
- Test Cases which are **time consuming**

The following category of test cases is not suitable for automation:

- **Test Cases that are newly designed** and not executed manually at least once
- **Test Cases for which the requirements are changing frequently**
- **Test cases** which are executed on **ad-hoc basis.**



Why Use Automated Testing Tools?

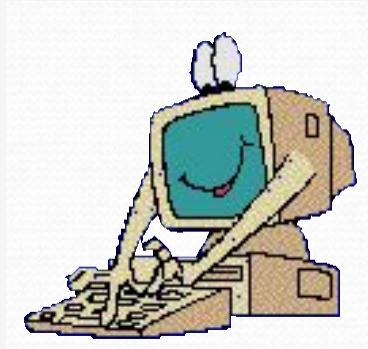


No Testing
Testing



Manual Testing

- **Time consuming**
- **Low reliability**
- **Human resources**
- **Inconsistent**

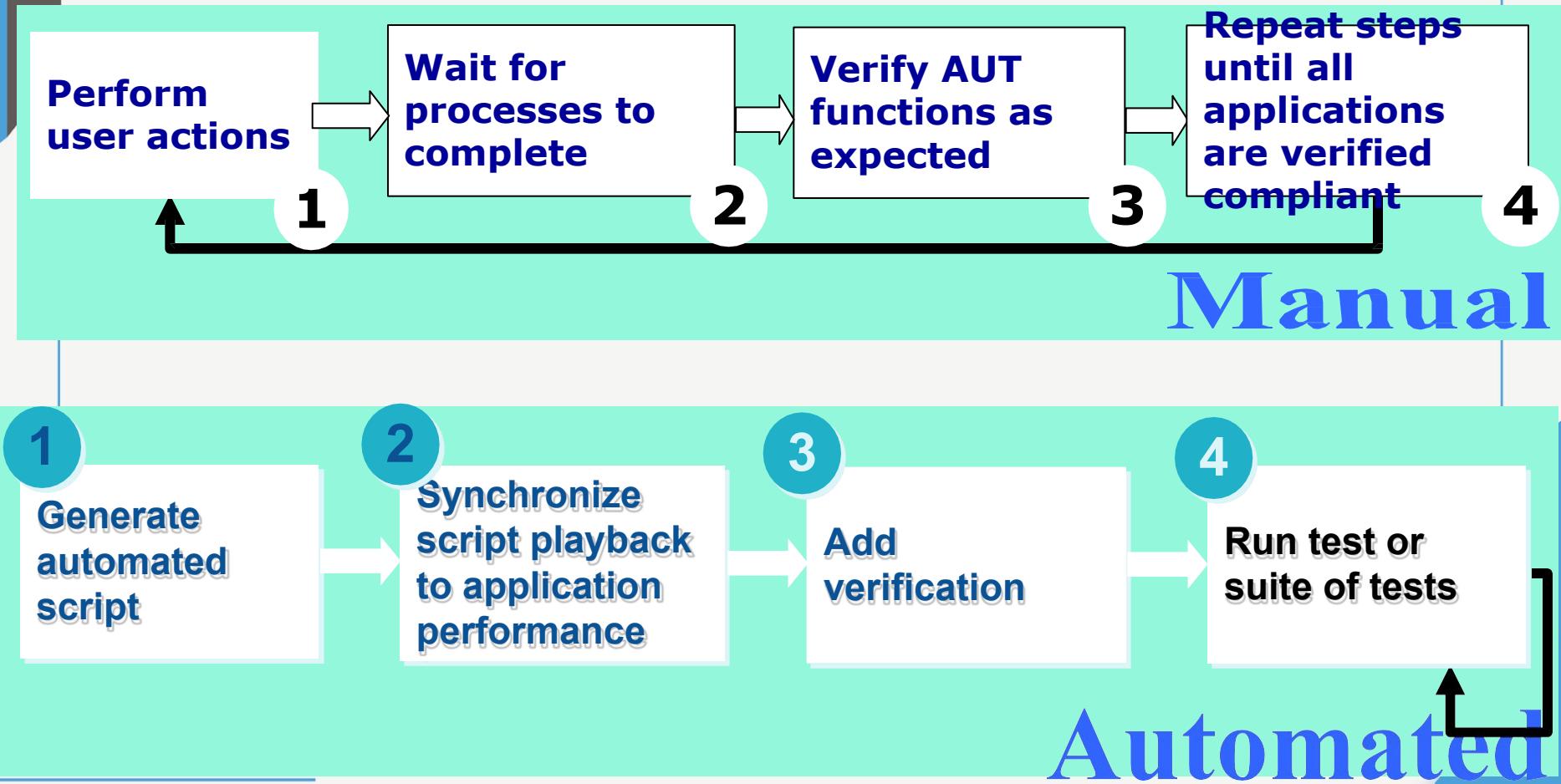


Automated Testing

Speed

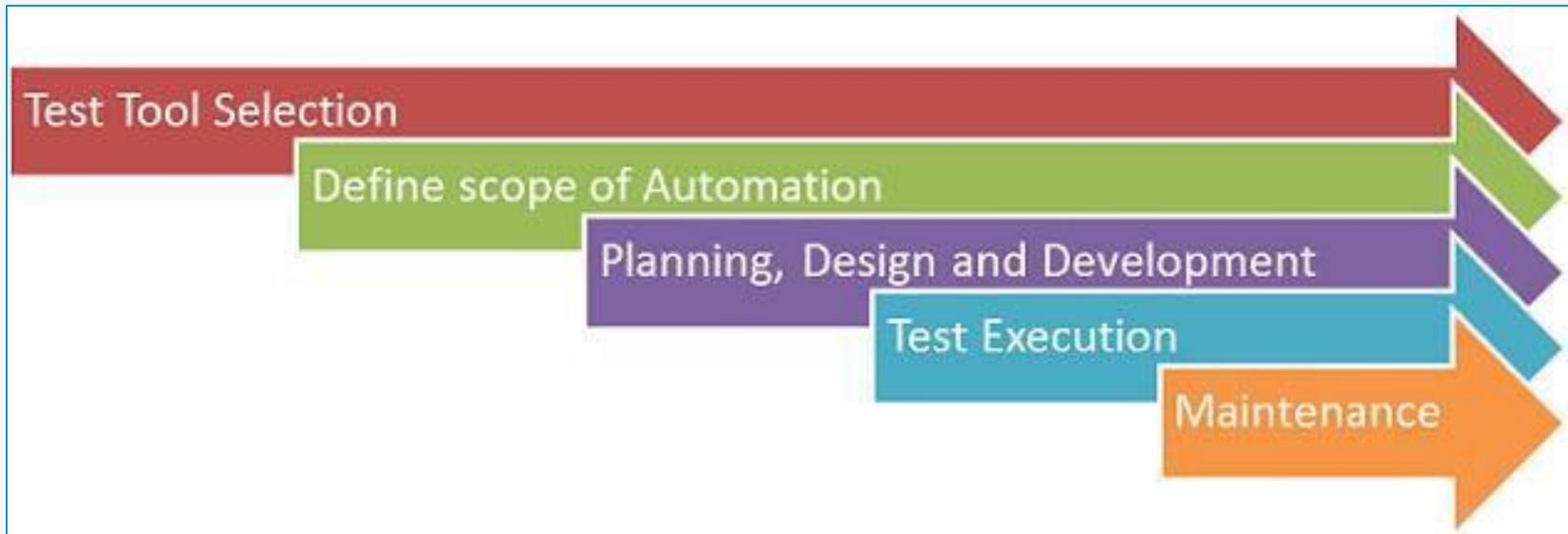
- **Repeatability**
- **Programming capabilities**
- **Coverage**
- **Reliability**
- **Reusability**

Manual To Automated Testing



Automation Process

- Test Tool Selection
- Define the Scope of Automation
- Planning, Designing and Development
- Test Estimation
- Maintenance



Framework in Automation

A **framework is set of automation guidelines** which help in

- Maintaining consistency of Testing
- Improves test structuring
- Minimum usage of code
- Less Maintenance of code
- Improve re-usability
- Non-Technical testers can be involved in code
- Training period of using the tool can be reduced
- Involves Data wherever appropriate

There are four types of framework used in software automation testing:

- Data Driven Automation Framework
- Keyword Driven Automation Framework
- Modular Automation Framework
- Hybrid Automation Framework

Benefits of Automation Testing

- 70% faster than the manual testing
- Wider test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Cost
- Improves accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation
- Early time to market

Automation Framework

What is Framework?

- A **test automation framework** is a set of assumptions, concepts and tools that provide support for automated software testing.
- The main advantage of such a framework is the low cost for maintenance.
- If there is change to any test case then only the test case file needs to be updated and the Driver Script and Startup script will remain the same.
- Ideally, there is no need to update the scripts in case of changes to the application.
- Choosing the right framework/scripting technique helps in maintaining lower costs.
- The costs associated with test scripting are due to development and maintenance efforts.
- The approach of scripting used during test automation has effect on costs.

What is Framework?

Instead of providing a bookish definition of a framework, let's consider an example:

- I am sure you have attended a seminar / lecture / conference where the participants was asked to observe the following guidelines -
- Participants should occupy their seat 5 minutes before start of lecture
- Bring along a notebook and pen for note taking.
- Read the abstract so you have an idea of what the presentation will be about.
- Mobile Phones should be set on silent
- Use the exit gates at opposite end to the speaker should you require to leave in middle of the lecture.

Load Runner up by HP(LR)

Agenda

- Why Performance ?
- Definitions: Performance Testing
- Benchmark Design
- Performance Testing Tools
- Load Runner Components
- What is load testing process?
- Getting Familiar with Mercury Tours
- Application Requirements

Some Testing Definitions

- **Stress Testing:** Stress Testing is done in order to check when the application fails by reducing the system resources such as RAM, HDD etc. and keeping the number of users as constant.

- **Load Testing:** Load Testing is done in order to check when the application fails by increasing the number of users and keeping the system resources as constant.

- **Performance Testing:** The term performance can mean measuring response time, throughput resource utilization, or some other system characteristic(or group them) by varying the number of users.

Benchmark Design

- The Benchmark is the representative workload used during the performance test run. It should be representative of the likely real-world operating conditions.

- Benchmark is provided by the client.

- In Industry scenario the benchmark is as follows:

- No. of transactions passed per second ≥ 8
- Response time ≤ 5 sec.

Performance Testing Tools

- Segue Silk Performer
- Rational Team Test
- Mercury Load Runner
- Empirix e-Load/RSW
- Soft Light Tools Loader

Seleniu m IDE

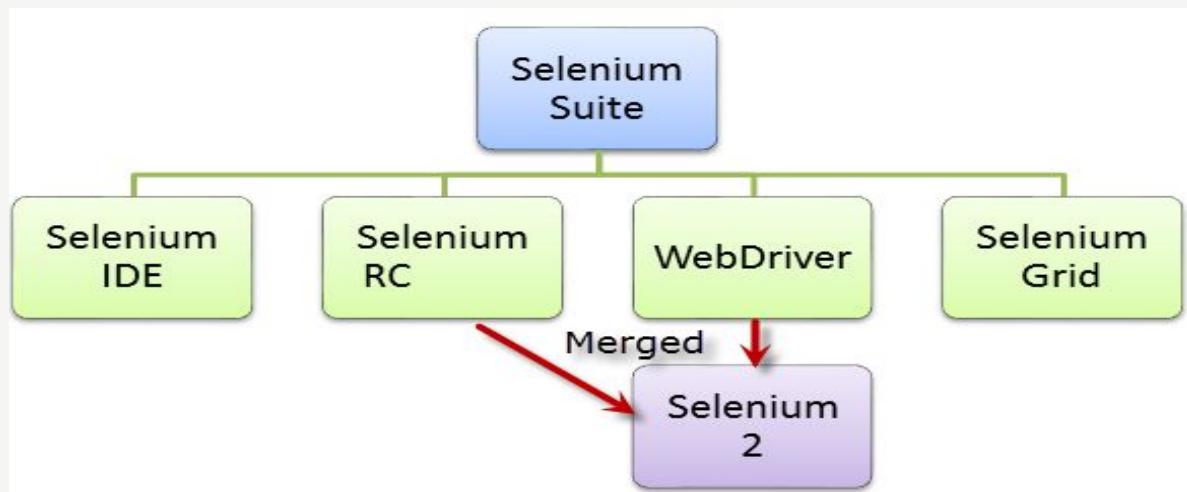
What is Selenium?

- Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms.

- It is quite similar to HP Quick Test Pro (QTP) only that Selenium focuses on automating web-based applications.

- Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. It has four components.

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- Web Driver
- Selenium Grid



Who developed Selenium?

Since Selenium is a collection of different tools, it had different developers as well. Below are the key persons who made notable contributions to the Selenium Project

- Primarily, Selenium was created by Jason Huggins in 2004.

- An engineer at **ThoughtWorks**, he was working on web application that required frequent testing.

- Having realized that the repetitious manual testing of their application was becoming more and more inefficient, he created a JavaScript program that would automatically control the browser's actions.

- He named this program as "**JavaScriptTestRunner**".



- Seeing potential in this idea to help automate other web applications, he made **JavaScriptRunner** open-source which was later renamed as **Selenium Core**.

Selenium RC

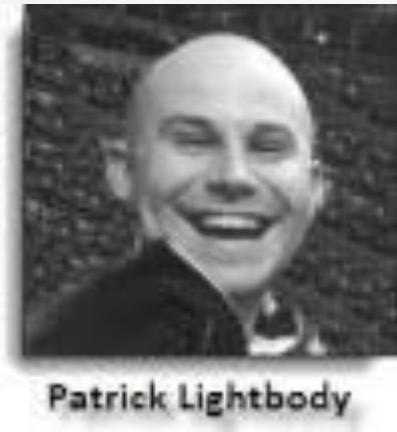
Unfortunately; testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the same origin policy. So another Thought Work's engineer, Paul Hammant , decided to create a server that will act as an HTTP proxy to “trick” the browser into believing that Selenium Core and the web application being tested come from the same domain. This system became known as the Selenium Remote Control or Selenium 1.



Paul Hammant

Selenium Grid

Selenium Grid was developed by Patrick Lightbody to address the need of minimizing test execution times as much as possible. He initially called the system “Hosted QA.” It was capable of capturing browser screenshots during significant stages, and also of sending out Selenium commands to different machines simultaneously.



Selenium IDE

Shinya Kasatani of Japan created Selenium IDE, a Firefox extension that can automate the browser through a record-and-playback feature. He came up with this idea to further increase the speed in creating test cases. He donated Selenium IDE to the Selenium Project in 2006.



Selenium Web Driver

Simon Stewart created WebDriver circa 2006 when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core. It was the first cross-platform testing framework that could control the browser from the OS level.



Birth of Selenium 2

In 2008, the whole Selenium Team decided to merge WebDriver and Selenium RC to form a more powerful tool called Selenium 2, with WebDriver being the core. Currently, Selenium RC is still being developed but only in maintenance mode. Most of the Selenium Project's efforts are now focused on Selenium 2.

So, Why the Name Selenium?

It came from a joke which Jason cracked one time to his team. Another automated testing framework was popular during Selenium's development, and it was by the company called Mercury Interactive (yes, the company who originally made QTP before it was acquired by HP). Since Selenium is a well-known antidote for Mercury poisoning, Jason suggested that name. His teammates took it, and so that is how we got to call this framework up to the present.

Introduction Selenium IDE

- Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite and is the easiest one to learn.
- It is a Firefox plugin that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a prototyping tool.
- If you want to create more advanced test cases, you will need to use either Selenium RC or WebDriver.

Pros & Cons Selenium IDE

Pros

- Very easy to use and install.
- No programming experience is required, though knowledge of HTML and DOM are needed
- Can export tests to formats usable in Selenium RC and WebDriver
- Has built-in help and test results reporting module.
- Provides support for extensions.

Cons

- Available only in Fire fox
- Designed only to create prototypes of tests.
- No support for iteration and conditional operations
- Test execution is slow compared to that of Selenium RC and WebDriver.

Introduction Selenium RC

- Selenium RC was the flagship testing framework of the whole Selenium project for a long time.

- This is the first automated web testing tool that allowed users to use a programming language they prefer.

- As of version 2.25.0, RC can support the following programming languages:

- Java
- C#
- PHP
- Python
- Perl
- Ruby

Pros & Cons Selenium RC

Pros

- Cross browser and cross platform
- Can perform looping and conditional operations
- Can support data-driven testing
- Has matured and complete API
- Can readily support new browsers
- Faster execution than IDE

Cons

- Installation is more complicated than IDE
- Must have programming knowledge
- Needs Selenium RC Server to be running
- API contains redundant and confusing commands
- Browser interaction is less realistic
- Inconsistent result & Users JavaScript
- Slower execution time than WebDriver.

Introduction Selenium Driver

- The WebDriver proves itself to be better than both Selenium IDE and Selenium RC in many aspects.
- It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for automation. It controls the browser by directly communicating to it.
- The supported languages are the same as those in Selenium RC.
 - Java
 - C#
 - PHP
 - Python
 - Perl
 - Ruby

Pros & Cons Selenium Driver

Pros

- Simpler installation than Selenium RC
- Communicates directly to the browser
- Browser integration is more realistic.
- No need for a separate component such as the RC Server
- Faster execution time than IDE and RC

Cons

- Installation is more complicated than Selenium IDE
- Requires programming knowledge
- Cannot readily support new browser
- Has no built-in mechanism for logging runtime message and generating test results

Introduction Selenium Driver

- Selenium Grid is a tool used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

- Features:

- Enables simultaneous running of tests in multiple browsers and environments.
- Saves time enormously.
- Utilizes the hub-and-nodes concept. The hub acts as a central source of Selenium commands to each node connected to it.

Features of Selenium Driver

- Enables simultaneous running of tests in multiple browsers and environments.
- Saves time enormously.
- Utilizes the hub-and-nodes concept. The hub acts as a central source of Selenium commands to each node connected to it.

Advantages of Selenium over QTP

Selenium	QTP
Open source, free to use, and free of charge.	Commercial.
Highly extensible	Limited add-ons
Can run tests across different browsers	Can only run tests in Firefox , Internet Explorer and Chrome
Supports various operating systems	Can only be used in Windows
Supports mobile devices	Supports mobile devise using 3 rd party software
Can execute tests while the browser is minimized to be visible on the desktop	Needs to have the application under test
Can execute tests in parallel.	Can only execute in parallel but using Quality Center which is again a paid product.

Advantages of QTP over Selenium

QTP	Selenium
Can test both web and desktop applications	Can only test web applications
Comes with a built-in object repository	Has no built-in object repository
Automates faster than Selenium because it is a fully featured IDE.	Automates at a slower rate because it does not have a native IDE and only third party IDE can be used for development
Data-driven testing is easier to perform because it has built-in global and local data tables.	Data-driven testing is more cumbersome since you have to rely on the programming language's capabilities for setting values for your test data
Can access controls within the browser(such as the Favorites bar, Address bar, Back and Forward buttons, etc.)	Cannot access elements outside of the web application under test
Provides professional customer support	No official user support is being offered.
Has native capability to export test data into external formats	Has no native capability to export runtime data onto external formats
Parameterization Support is in built	Parameterization can be done via programming but is difficult to implement.
Test Reports are generated automatically	No native support to generate test /bug reports.