

# Software Testing

## Hand Book

TOPS Technologies

## 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?

**“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

**“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
- 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.

### • 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.
- 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 a 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 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

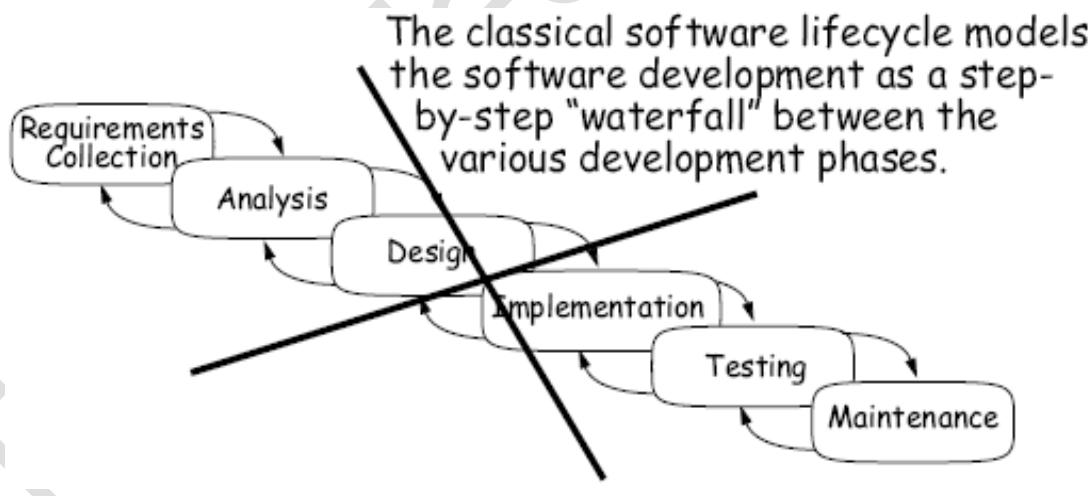
**In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “maintenance”!**

- 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 Engineering Models / Methodologies**

### **Waterfall Model (Classical Software Cycle)**

- 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.

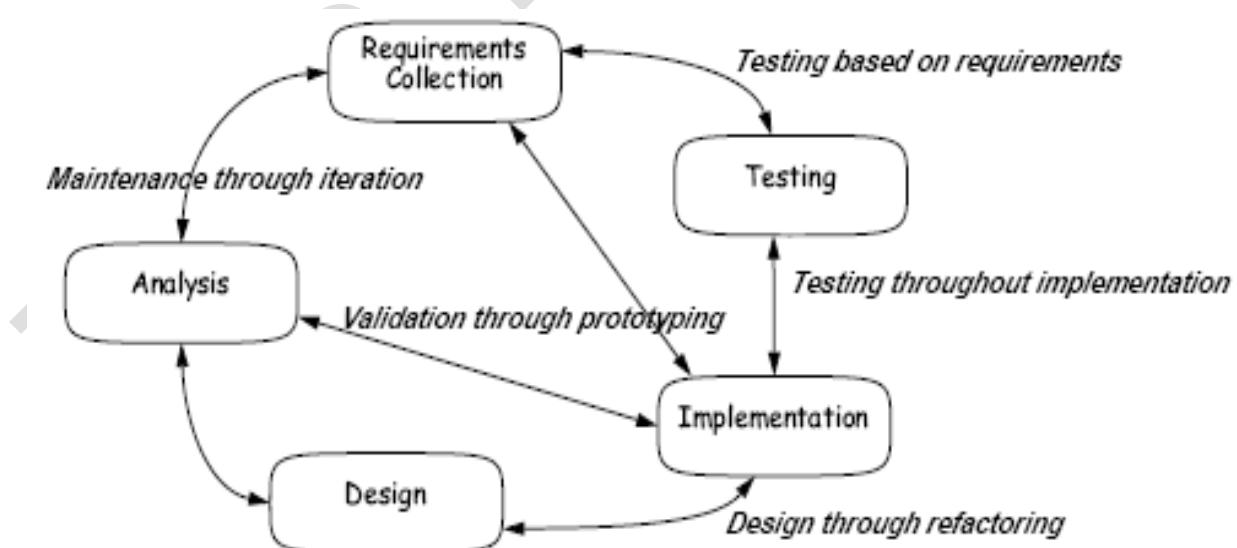
## Pros (Why Waterfall Model)

- 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)

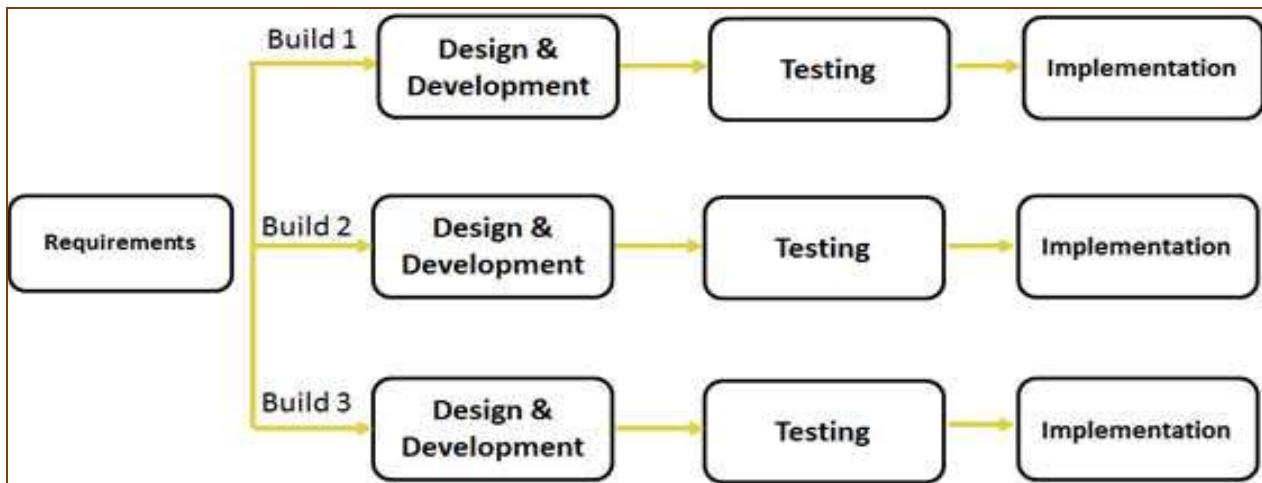
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. 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.
- 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.

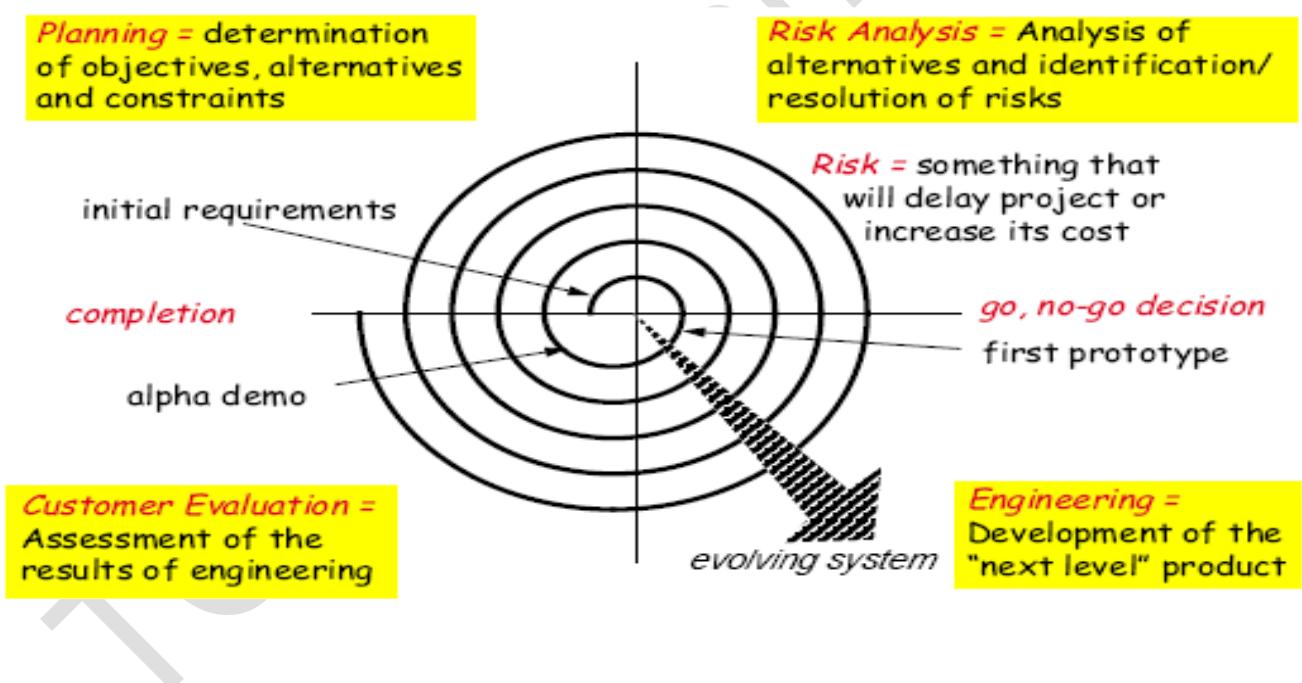
- It supports changing requirements.
- Initial Operating time is less.

- 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/Methodology



## 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.

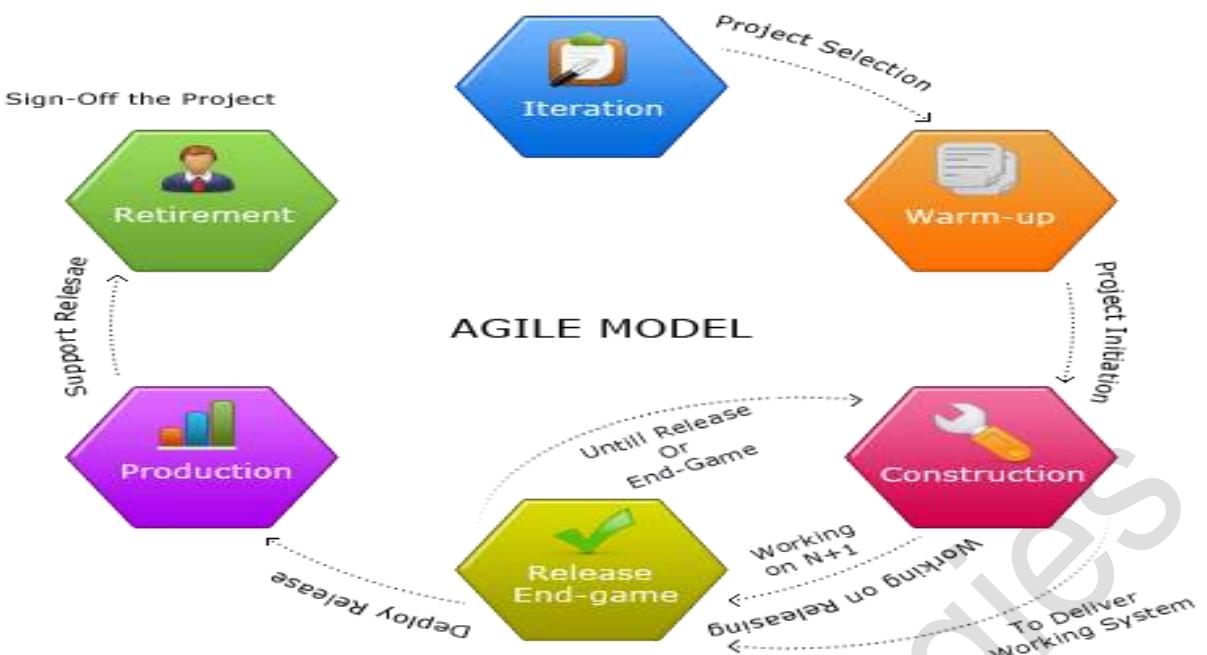
## Cons (Why It doesn't work)

- 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.

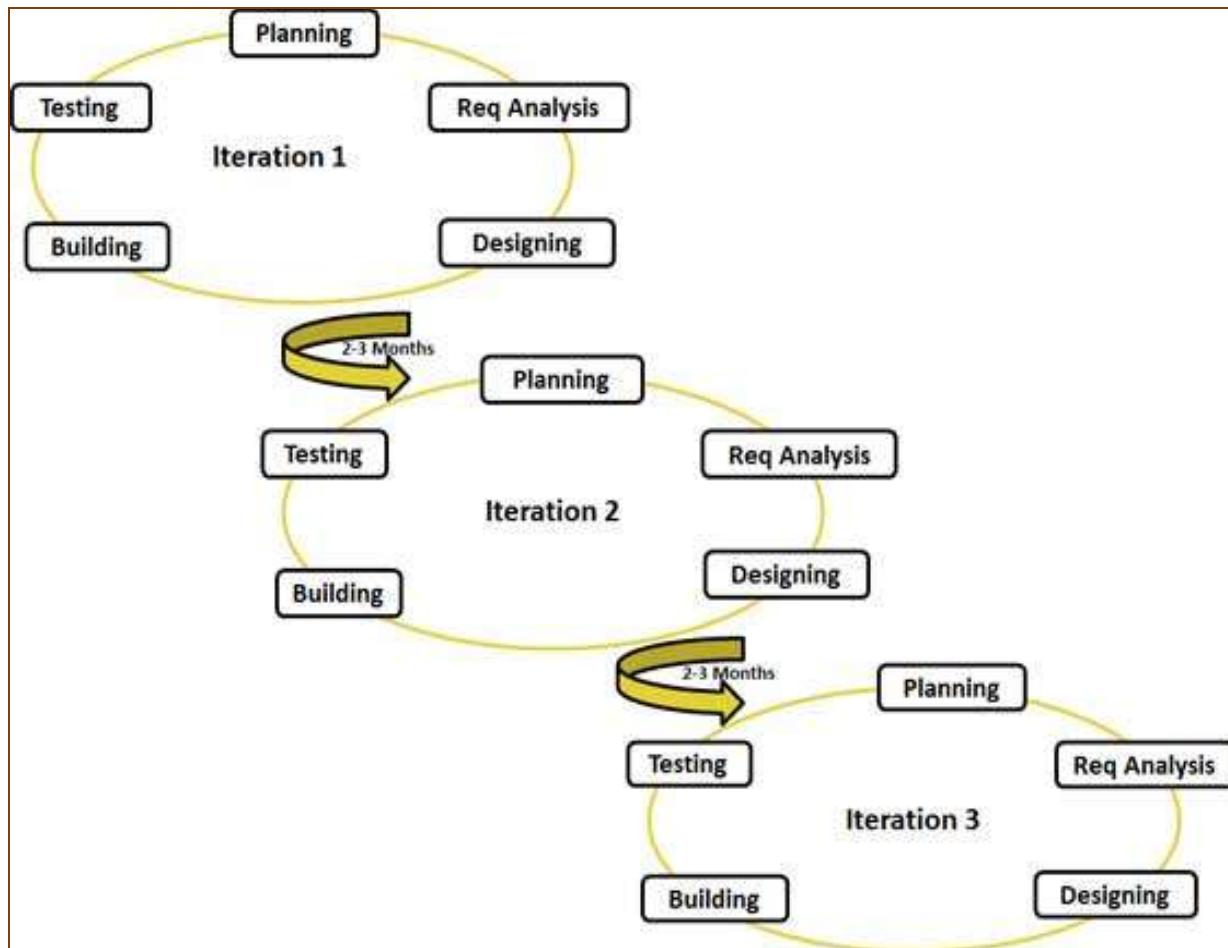
## Agile Model/Methodology

- 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.

## Agile Model Work Flow



# Agile Model Work Flow



## Agile Manifesto Principles

- **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** - Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
- **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements..
- **Responding to change** - agile development is focused on quick responses to change and continuous development.

## 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

## 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.
- A use case is a technique for documenting the potential requirements of a new system or software change.
- Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal.
- Use cases typically avoid technical words, preferring instead the language of the end user or domain expert. Use cases are often coauthored by requirements engineers and stakeholders.
- Use cases do not describe any internal workings of the system, nor do they explain how that system will be implemented.
- They simply show the steps that a user follows to perform a task.
- All the ways that users interact with a system can be described in this manner.

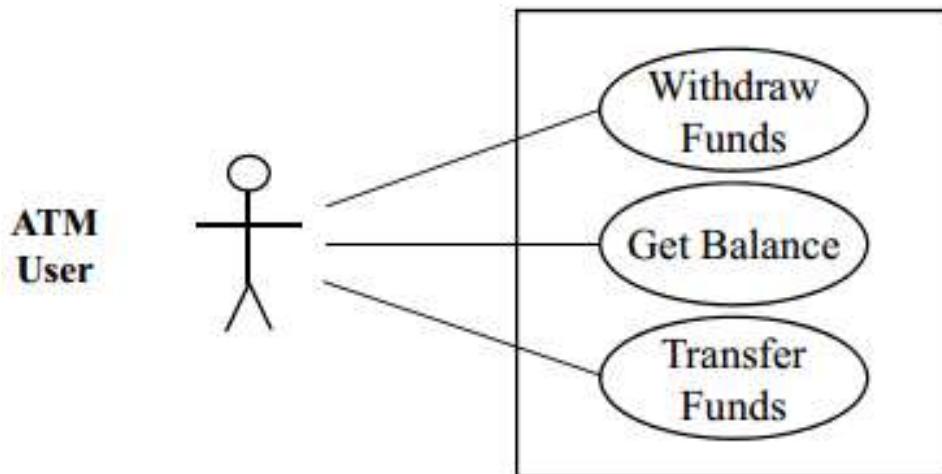
## Viewpoints

- Stakeholders represent different problem view points
  - Interview as many different kinds of stakeholders as possible/necessary
  - Translate requirements into use-cases or “stories” about the desired system involving a fixed set of actors(users and system)

**objects)**

- For each use-case, capture **both typical and exceptional usage scenarios**
- Users tends to think about systems in term of “features”
  - You must get them to tell you **stories (use-case)** involving those features.
  - Use-cases and scenarios can tell you if the requirements are **complete and consistent!**

## Example of ATM



- **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**
- Use cases are commonly elaborated (or documented)
- Elaboration is first written textually
  - Details of operation
  - Alternatives model choices and conditions during execution

## Elaboration of ATM Use Case

- **Use Case Name :** Withdraw Funds
- **Summary:** Customer uses a valid card to withdraw funds from a valid bank account.
- **Actor :** ATM Customer
- **Precondition :** ATM is displaying the idle welcome message : ATM is displaying the idle welcome message

### Description :

- Customer inserts an ATM Card into the ATM Card Reader.
- If the system can recognize the card, it reads the card number.
- System prompts the customer for a PIN.
- Customer enters PIN.
- System checks the card's expiration date and whether the card has been stolen or lost.
- If the card is valid, the system checks if the entered PIN matches the card PIN.
- If the PINs match, the system finds out what accounts the card can access.
- System displays customer accounts and prompts the customer to choose a type of

transaction. There are three types of transactions, Withdraw Funds, Get Balance and Transfer Funds. (The previous eight steps

are part of all three use cases; the following steps are unique to the Withdraw Funds use case.

- Customer selects Withdraw Funds, selects the account number, and enters the amount.
- System checks that the account is valid, makes sure that customer has enough funds in the account makes sure that the daily limit has not been exceeded and funds in the account, makes sure that the daily limit has not been exceeded, and checks that the ATM has enough funds.
- If all four checks are successful, the system dispenses the cash.
- System prints a receipt with a transaction number, the transaction type, the amount withdrawn, and the new account balance.
- System ejects card.
- System displays the idle welcome message.

## Alternatives:

- If the system cannot recognize the card, it is ejected and the welcome message is displayed.
- If the current date is past the card's expiration date, the card is confiscated and the welcome message is displayed.
- If the card has been reported lost or stolen it is confiscated and the welcome message is if the card has been reported lost or stolen, it is confiscated and the welcome message is displayed.
- If the customer entered PIN does not match the PIN for the card, the system prompts for new PIN.
- If the customer enters an incorrect PIN three times, the card is confiscated and the welcome message is displayed.
- If the account number entered by the user is invalid, the system displays an error message, ejects the card and the welcome message is displayed.
- If the request for withdraw exceeds the maximum allowable daily withdrawal amount, the system displays an apology message, ejects the card and the welcome message is displayed.
- If the request for withdraw exceeds the amount of funds in the ATM, the system displays an apology message, ejects the card and the welcome message is displayed.
- If the customer enters Cancel, the system cancels the transaction, ejects the card and the welcome message is displayed.

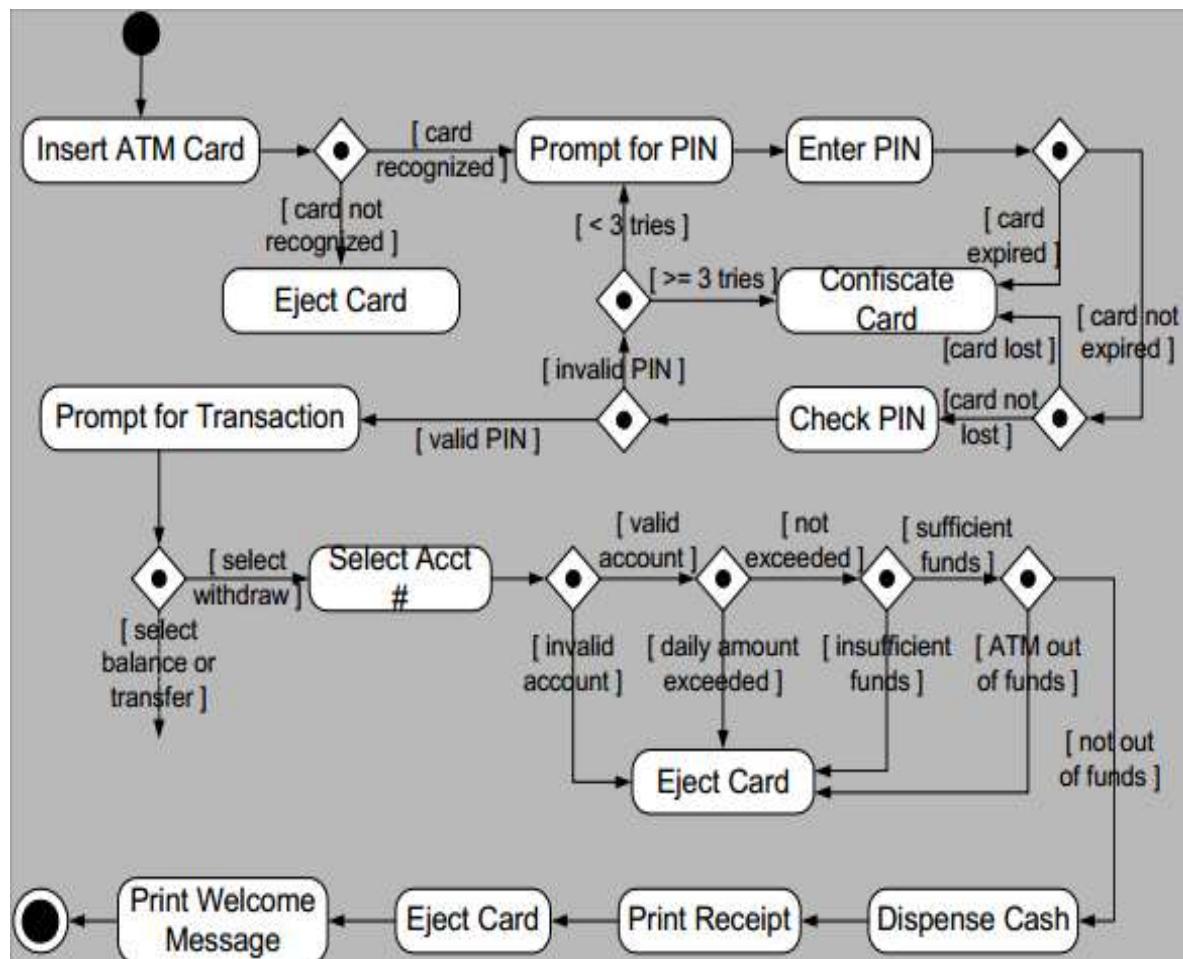
## Post condition :

Funds have been withdrawn from the customer's account.

## Use Cases to Activity Diagrams

- Activity diagrams indicate flow among activities
- Activities should model user level steps
- Two kinds of nodes: Two kinds of nodes:
  - Action states
  - Sequential branches
- Use case descriptions become action state nodes in the activity diagram
- Alternatives are sequential branch nodes
- Flow among steps are edges
- Activity diagrams usually have some helpful characteristics:
  - Few loops
  - Simple predicates
  - No obvious DU pairs

# ATM Withdraw Activity Graph



# Covering Activity Graph

- Node Coverage
  - Inputs to the software are derived from labels on nodes and predicates
  - Used to from test case values
- Edge Coverage
- Data flow techniques do not apply
- Scenario Testing
  - **Scenario : A complete path through a use case activity graph**
  - Should make semantic sense to the users
  - Number of paths often finite
  - If not, scenarios defined based on domain knowledge
  - Use “specified path coverage”, where the set S of paths is the set of scenarios
  - Note that specified path coverage does not necessarily subsume edge coverage, but scenarios should be defined so that it does

# Software Requirements Specification

# Introduction

- 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. Nonfunctional requirements can be divided into following categories:
  - Usability
  - Reliability
  - Performance
  - Security

## Product & Project Based Application

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.

TOPS Technologies

# Object Oriented Programming

## 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 practice 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 color, 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).
- 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.

# Example: The Animal class

```

public class Animal
{
    private String animalName;
    private String animalType;
    public Animal()
    {
        System.out.println("in default constructor ");
    }

    public Animal(String animalName, String animalType)
    {
        this.animalName = animalName;
        this.animalType = animalType;
    }

    public String getAnimalName()
    {
        return animalName;
    }

    public void setAnimalName(String animalName)
    {
        this.animalName = animalName;
    }

    public String getAnimalType()
    {
        return animalType;
    }

    public void setAnimalType(String animalType)
    {
        this.animalType = animalType;
    }
}

public static void main(String args[])
{
    Animal a = new Animal("Cow", "Mammal");
    System.out.println("name : " + a.animalName);
    System.out.println("type : " + a.animalType);
    Animal a1 = new Animal();
    a1.setAnimalName("Dog");
    a1.setAnimalType("Mammal");
    System.out.println(" name : " + a1.getAnimalName());
    System.out.println("type : " + a1.getAnimalType());
}

```

# 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 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**.

## Abstraction Example

```

package com.abstractdemo;
public class MatrixImpl extends Matrix
{
    @Override
    public int[][] add(int[][] a, int[][] b)
    {
        // TODO Auto-generated method stub
        int[][] c = new int[a.length][b.length];
        for (int i = 0; i < a.length; i++)
        {
            for (int j = 0; j < b.length; j++)
            {
                c[i][j] = a[i][j] + b[i][j];
            }
        }
        return c;
    }
}

```

# Abstraction Example

```
Demo.java
System.out.println("Enter a values : ");
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        a[i][j] = scanner.nextInt();
    }
}
System.out.println("Enter b values : ");
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        b[i][j] = scanner.nextInt();
    }
}
MatrixImpl impl = new MatrixImpl();
c = impl.add(a, b);
System.out.println("addition:::: ");
for (int i = 0; i < c.length; i++)
{
    for (int j = 0; j < c.length; j++)
    {
        System.out.print(" " + c[i][j]);
    }
    System.out.println("\n");
```

# 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 behavior 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.



- Poly refers too 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 existing 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.

## Overloading Example

```

class DemoOverload
{
    public int add(int x, int y)
    {
        return x+y;
    }
    public int add(int x, int y, int z)
    {
        return x + y + z;
    }
    public int add(double x, int y)
    {
        return (int) x + y;
    }
    public int add(int x, double y)
    {
        return x + (int) y;
    }
}

public class Test
{
    public static void main(String[] args)
    {
        DemoOverload demo=new DemoOverload();
        System.out.println(demo.add(2,3));
        System.out.println(demo.add(2,3,4));
        System.out.println(demo.add(2,3,4));
        System.out.println(demo.add(2.5,3));
    }
}

```

# Overriding

- Overriding is defining a method in a subclass with the same name and type signature as a method in its super class and when this subclass instance appears in the super class context like

**Parent p = new Child()**

- and when we execute an overridden method as **p.myMethod()**, the subtype's version of that method is executed.
- Here, the actual method called will depend on the object at runtime, not the reference type.
- Consider an example class Shape with a **draw()** method.
- It can have subclasses Circle and Square.
- An object of Circle or Square can be assigned to a Shape reference as

**Shape s = new Circle();**

- While executing **draw()** on the Shape reference, it will draw a Circle or Square based on the actual object assigned to it at runtime.

## Overriding Example

```

class Vehicle
{
    public void move()
    {
        System.out.println("Vehicles can move!!");
    }
}

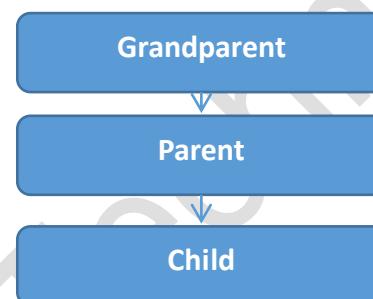
class MotorBike extends Vehicle
{
    public void move()
    {
        System.out.println("MotorBike can move and
accelerate too!!");
    }
}

public class TestOverriding
{
    public static void main(String[] args)
    {
        Vehicle vh = new MotorBike();
        vh.move(); // prints MotorBike can move and
        accelerate too!!
        vh = new Vehicle();
        vh.move(); // prints Vehicles can move!!
    }
}

```

# 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.
- 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.



- 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**

# Inheritance Example

Shape.java

```
public class Shape
{
    public double getArea()
    {
        return 0.5*base*height;
    }
    } private String color;
    // Constructor
    public Shape (String color)
    {
        this.color = color;
    }
    public String toString()
    {
        return "Shape of color=\\" + color + "\\";
    }
    public double getArea()
    {
        System.out.println("Shape unknown! Cannot compute area!");
        return 0;
    }
}
```

Rectangle.java

```
public class Rectangle extends Shape
{
    int length;
    int width;
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }
    public String toString()
    {
        return "Rectangle of length=" + length + " and width=" + width + ",\n"
            + "subclass of " + super.toString();
    }
}
```

```
public double getArea()
{
    return length*width;
}
```

}

Triangle.java



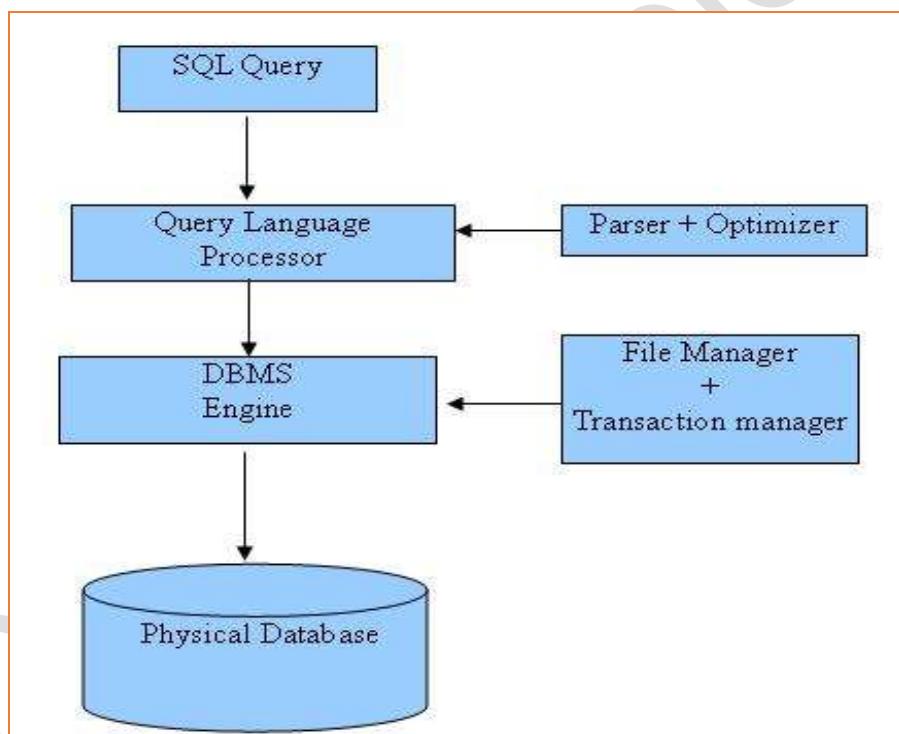
```
public class Triangle extends Shape
{
    private int base;
    private int height;
    public Triangle(String color, int base, int height)
    {
        super(color);
        this.base = base;
        this.height = height;
    }

    public String toString()
    {
        return "Triangle of base=" + base + " and height=" + height + ", subclass of
"
               +
               super.toString();
    }
}
```

# Database and Structure Query Language

## 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...”**

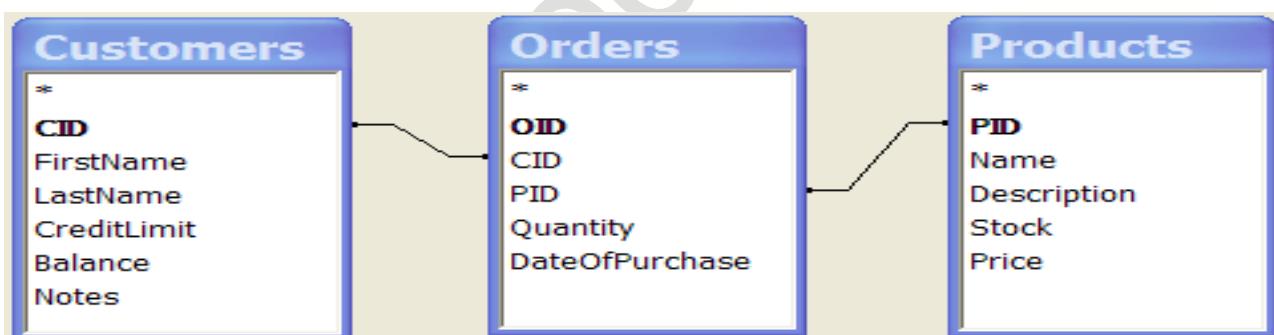


# Relational Databases

- 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...

Customers : Table						
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		

# 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: [◀◀] [◀] [▶] [▶▶] [✳] 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	

Record: [◀◀] [◀] [▶] [▶▶] [✳] of 5

# Database Management Systems

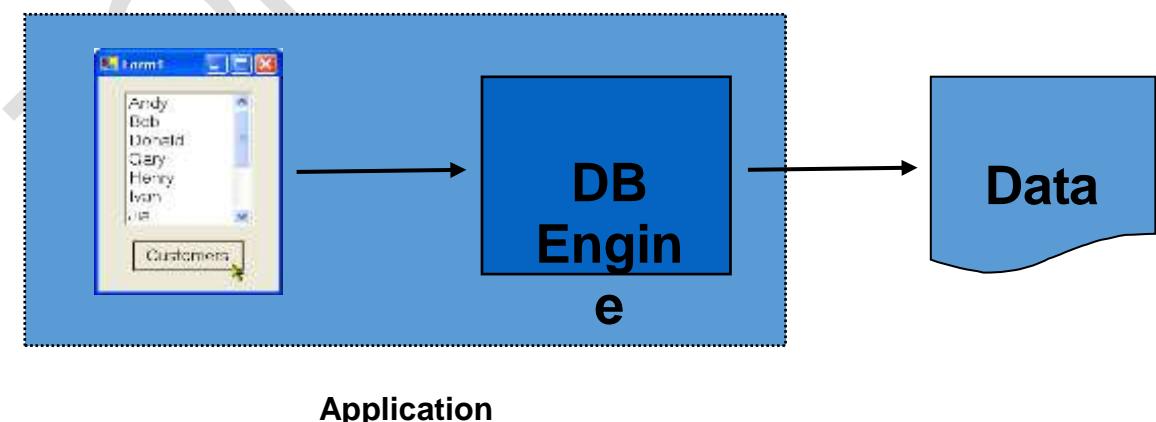
- A DBMS consists of 2 main pieces:
  - the data
  - the DB engine
  - the data is typically stored in one or more files



- Two most common types of DBMS are:
  - Local
  - Server

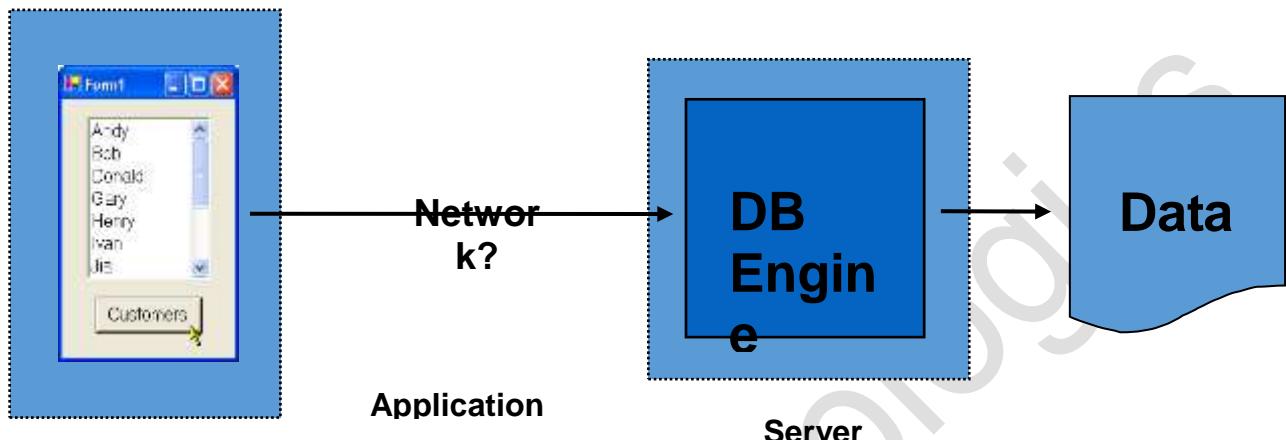
## Local DBMS

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



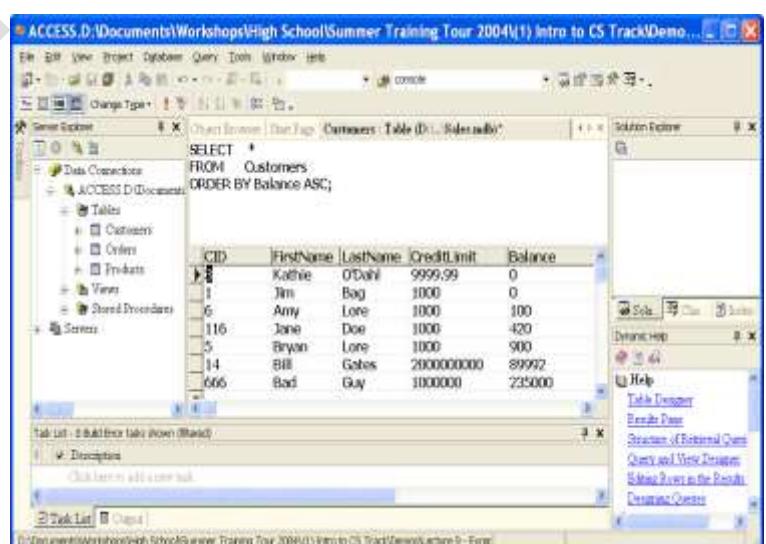
# Server DBMS

- 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
  - You 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, and 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
- 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
<b>CREATE</b>	Creates a new table, a view of a table, or other object in database
<b>ALTER</b>	Modifies an existing database object, such as a table.
<b>DROP</b>	Deletes an entire table, a view of a table or other object in the database.

# DQL – Data Query Language

Command	Description
<b>SELECT</b>	Retrieves certain records from one or more tables

# DML – Data Manipulation Language

Command	Description
<b>INSERT</b>	Creates a record
<b>UPDATE</b>	Modifies records
<b>DELETE</b>	Deletes records

# DCL – Data Control Language

Command	Description
<b>GRANT</b>	Gives a privilege to user
<b>REVOKE</b>	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_ype};
```

- **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**

 **TOPS** Technologies    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;

- **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};

- **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 (arithematicfunction 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_field = 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_field = 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
- Static Testing and Techniques
- Software Development Models
- Defect Management and Tracking
- Test Organization and Management
- Psychology of Testing
- Agile Testing

## Fundamentals 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.
- **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
- 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.'**
- 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.
- 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.

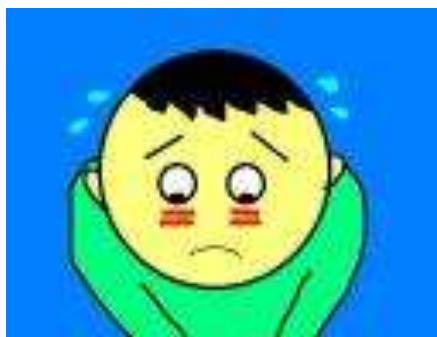
## Why Testing is Necessary?

- 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 mistaken, 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.
- 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!





- **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;



## 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

## 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 Principles of Testing

### General Testing Principles

- Testing shows presence of Defects
- Exhaustive Testing is Impossible!
- Early Testing
- Defect Clustering
- The Pesticide Paradox
- Testing is Context Dependent

- Absence of Errors Fallacy

## Testing shows presence of Defects

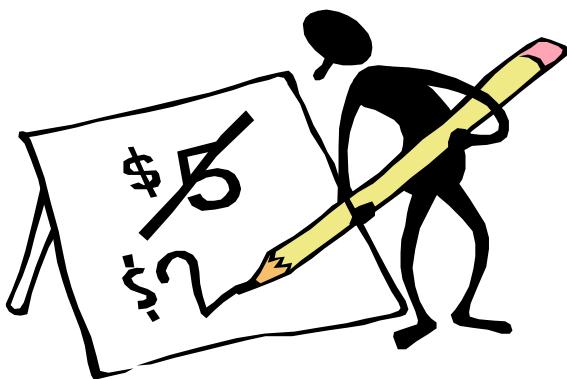
- Testing can show that defects are present, but cannot prove that there are no defects.
- Testing **reduces the probability of undiscovered defects** remaining in the software but, even if no defects are found, it is not a proof of correctness.
- We test to find Faults
- As we find more defects, the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

## 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 (515) 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



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**

$$20 \times 4 \times 3 \times 10 \times 2 \times 100 = 480,000 \text{ tests}$$

**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**

## 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



# 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
- 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



# Testing is 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'

## Error, Bugs, Defects

## 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

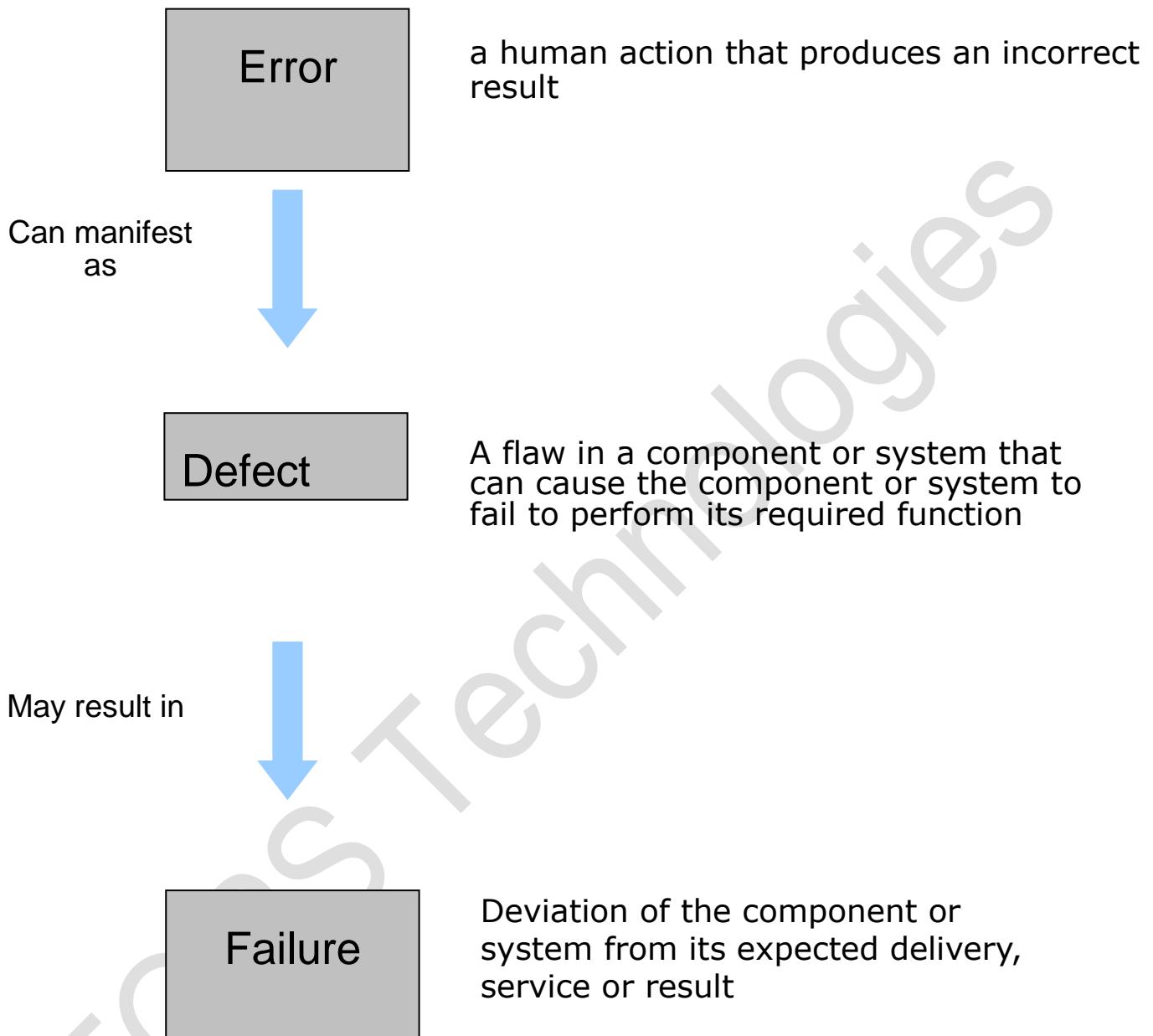




## Errors, Defects and Failures

- 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.
- **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

# 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.**"
- 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.

# Risk, Types of risks

## 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 Organization

## Introduction

- The effectiveness of finding defects by testing and reviews can be improved by using independent testers. Options for independence are:
  - No independent testers. Developers test their own code.
  - Independent testers within the development teams.
  - Independent test team or group within the organization, reporting to project management or executive management.
  - Independent testers from the business organization or user community.
  - Independent test specialists for specific test targets such as usability testers, security testers or certification testers (who certify a software product against standards and regulations).
  - Independent testers outsourced or external to the organization.
  - For large, complex or safety critical projects, it is usually best to have multiple levels of testing, with some or all of the levels done by independent testers. Development staff may participate in testing, especially at the lower levels, but their lack of objectivity often limits their effectiveness.
- The independent testers may have the authority to require and define test processes and rules, but testers should take on such process-related roles only in the presence of a clear management mandate to do so.
- **The benefits of independence include:**
  - Independent testers see other and different defects, and are unbiased.
  - An independent tester can verify assumptions people made during specification and implementation of the system.
- **Drawbacks include:**
  - Isolation from the development team (if treated as totally independent).
  - Independent testers may be the bottleneck as the last checkpoint.
  - Developers may lose a sense of responsibility for quality.

# 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 difference 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

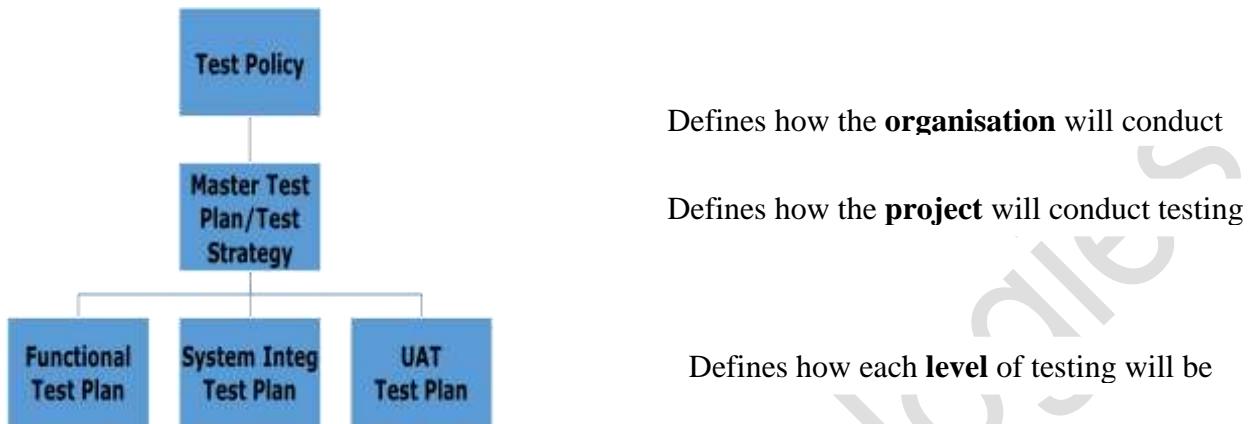


# Test Planning

- *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 organization'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

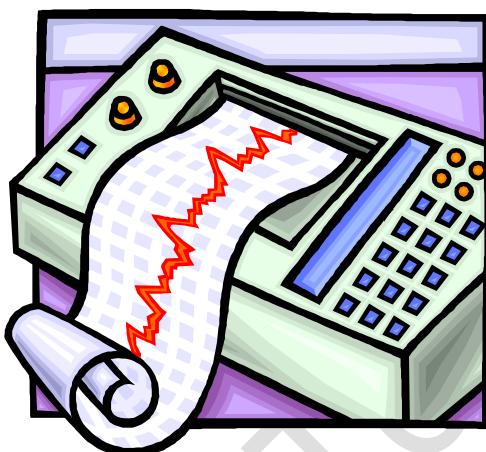
- **Approach:** Defining the overall approach of testing (the test strategy), including the definition of the test levels and entry and exit criteria.
- **Integrating and coordinating the testing activities into the software life cycle activities:** acquisition, supply, development, operation and maintenance.
- 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.

# Test Progress Monitoring – Why?

- Need to know the status of the testing project at any given point in time
- Need to provide visibility on the status of testing to other stake holders
  - Need to be able to measure your testing against your defined exit criteria
  - Need to be able to assess progress against
  - Planned schedule
  - Measure how you are tracking against your defined budget

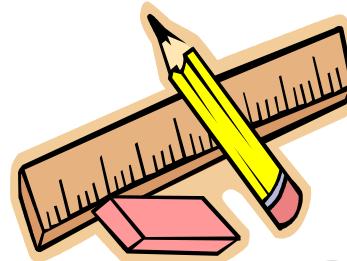


# Test Progress Monitoring – How?

- Typical Metrics include
  - **Test preparation** - Percentage of work done in test case preparation (or percentage of planned test cases prepared).
  - **Test environment preparation** - Percentage of work done in test environment preparation.
  - **Test case execution** (e.g. number of test cases run/not run, and test cases passed/failed).
  - **Defect statistics** (e.g. defect density, defects found and fixed, failure rate, and retest results).
  - **Test coverage** - e.g. % of requirements tested, risks or code coverage.
  - **Dates of test milestones** – monitoring the test schedule
  - Testing costs, including the cost compared to the benefit of finding the next defect or to run the next test.
  - Subjective confidence of testers in the product.

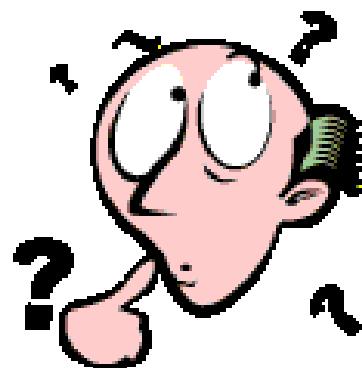
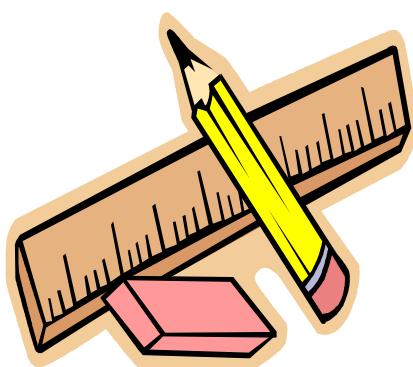
# Test Estimation

- Two approaches for the estimation of test effort are covered:
- The metrics-based approach: estimating the testing effort based on metrics of former or similar projects or based on typical values.
- The expert-based approach: estimating the tasks by the owner of these tasks or by experts.
- Once the test effort is estimated, resources can be identified and a schedule can be drawn up.
  - The testing effort may depend on a number of factors, including: Characteristics of the product: the quality of the specification and other information used for test models (i.e. the test basis), the size of the product, the complexity of the problem domain, the requirements for reliability and security, and the requirements for documentation.
  - Characteristics of the development process: the stability of the organization, tools used, test process, skills of the people involved, and time pressure.
  - The outcome of testing: the number of defects and the amount of rework required.



## Test Estimation - How?

- Estimates provide predictions of effort and cost for conduct of testing activities that support :
  - establishment of budgets
  - procurement of resources
  - production of a schedule for use in tracking and control
- by definition estimates are not exact and good estimates are ones that provide predictions within realistic tolerance limits



# Test Estimation - How?

- Estimation based on metrics from similar or previous projects or typical values
- Use metrics such as
  - Length of time in preparation
  - Length of time in execution
  - Defect turnaround time
  - Down time and delays
- However it is important that you compare metrics from similar sized projects.
- Size of project is also required
  - Function point analysis
  - LOC (lines of code)
- Estimating tasks by the owner of these tasks or by experts
- Experts may use formal metrics
- May be more gut feel and experience based



# Test Estimation Factors

- The testing effort can depend on:
  - Characteristics of the product:
    - the quality of the specification
    - the size of the product
    - the complexity of the problem domain
  - Characteristics of the development process:
    - the stability of the organization
    - tools used
    - test process
    - skills of the people involved
    - time pressure
  - The outcome of testing
    - the number of defects
    - The amount of rework required.
- Once estimation is complete the resource requirements and projects schedules can be finalized

# Test Reporting

- Test reporting is concerned with summarizing information about the testing endeavor, including:
  - What happened during a period of testing, such as dates when exit criteria were met?
  - Analyzed information and metrics to support recommendations and decisions about future
- Actions, such as an assessment of defects remaining, the economic benefit of continued testing, outstanding risks and the level of confidence in tested software.
  - The outline of a test summary report is given in ‘Standard for Software Test Documentation’ (IEEE 829).
  - Metrics should be collected during and at the end of a test level in order to assess:
  - The adequacy of the test objectives for that test level.
  - The adequacy of the test approaches taken.
  - The effectiveness of the testing with respect to its objectives.



## 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.
---	--	--	---

## When to Start Testing?

- An early start to testing reduces the cost, time to rework and error free software that is delivered to the client.
- However in Software Development Life Cycle (SDLC) testing can be started from the Requirements Gathering phase and lasts till the deployment of the software.
- However it also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested.
- Testing is done in different forms at every phase of SDLC like during **requirement gathering phase, the analysis and verifications of requirements are also considered testing.**
- Reviewing the design in the design phase with intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

## When to Stop Testing?

- Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested. Following are the aspects which should be considered to stop the testing:
  - Testing Deadlines.
  - Completion of test case execution.
  - Completion of Functional and code coverage to a certain point.
  - Bug rate falls below a certain level and no high priority bugs are identified.
  - Management decision.

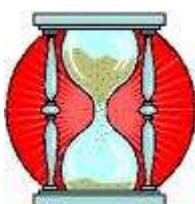


# How much Testing is enough?

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



- 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

- The level of formality depends on the context of the testing, including the organization, the maturity of testing and development processes, time constraints and the people involved.
- During test analysis, the test basis documentation is analyzed in order to determine what to test, **i.e.** to identify the test conditions.
  - A test condition is defined as an item or event that could be verified by one or more test cases (e.g. a function, transaction, quality characteristic or structural element).
- **Establishing traceability from test conditions** back to the specifications and requirements enables both impact analysis, when requirements change, and requirements coverage to be determined for a set of tests.
- During test analysis the detailed test approach is implemented to select the test design techniques to use, based on, among other considerations, the risks identified.
- During test design **the test cases and test data** are created and specified.
- **A test case consists of a set of input values, execution preconditions, expected results and execution post-conditions, developed to cover certain test condition(s).**
- The ‘Standard for Software Test Documentation’ (IEEE 829) describes the content of test design specifications (containing test conditions) and test case specifications.

# Introduction (Cont....)

- Expected results should be produced as part of the specification of a test case and include outputs, changes to data and states, and any other consequences of the test.
- If expected results have not been defined then a plausible, but erroneous, result may be interpreted as the correct one.
- **Expected results should ideally be defined prior to test execution.**
- During test implementation the test cases are developed, implemented, prioritized and organized in the test procedure specification.
- **The test procedure (or manual test script) specifies the sequence of action for the execution of a test.**
- If tests are run using a test execution tool, the sequence of actions is specified in a test script (which is an automated test procedure).
- **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

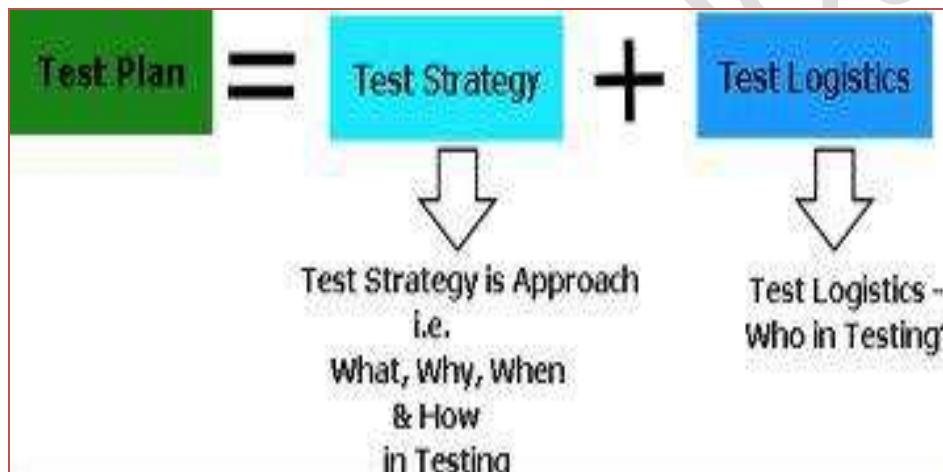
- Test analysis is the process of looking at something that can be used to derive test information. This basis for the tests is called the **test basis**.
- The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which 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.
- 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.

# Test Analysis (Cont....)

- As we know that testing everything is an impractical goal, which is known as exhaustive testing. We cannot test everything we have to select a subset of all possible tests.
- In practice the subset we select may be a very small subset and yet it has to have a high probability of finding most of the defects in a system. Hence we need some intelligent thought process to guide our selection called **test techniques**.
- Test conditions are documented in the IEEE 829 document called a **Test Design Specification**.

# Test Plan

- It is a high level document in which how to perform testing is described. The Test Plan document is usually prepared by the Test Lead or Test Manager and the focus of the document is to describe what to test, how to test, when to test and who will do what test.



- **Master test plan:** A test plan that typically addresses multiple test levels.
- **Phase test plan:** A test plan that typically addresses one test phase.
- A test plan will include the following.
  - Introduction to the Test Plan document
  - Assumptions when testing the application
  - List of test cases included in Testing the application
  - List of features to be tested
  - What sort of Approach to use when testing the software
  - List of Deliverables that need to be tested
  - The resources allocated for testing the application
  - Any Risks involved during the testing process
  - A Schedule of tasks and milestones as testing is started

# Test Strategy

- Test strategy is an outline that describes the testing portion of the software development cycle.
- It is created to inform QA, testers and developers about some key issues of the testing process.
- This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.

# High Level Requirement

#	Name	Description
	<b>Search</b>	
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
	<b>User account</b>	
200	Login	
201	Logout	
202	Forgot user name and password	
203	Create user account	

# Test Script

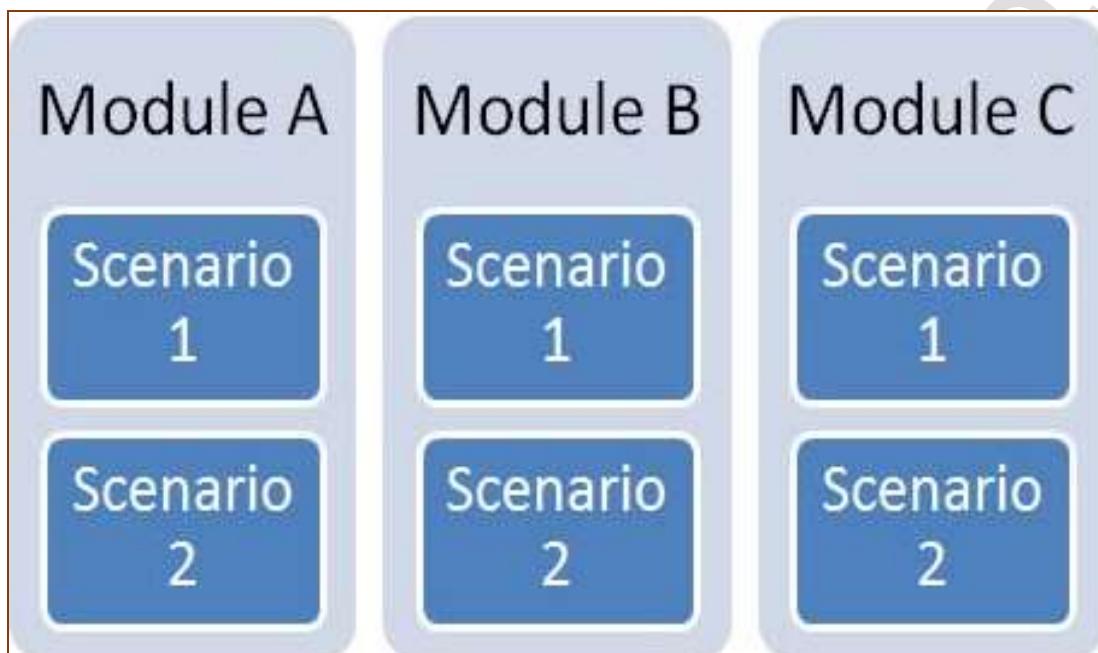
- A set of sequential instruction that detail how to execute a core business function
- One script is written to explain how to simulate each business scenario
- Written to a level of detail for which someone else (other than the script writer) would be able to easily execute
- Identifies the test condition 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
- 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.

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

TOPS Technologies

# 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.
- 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)
  - 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
- 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 analyzed 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 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

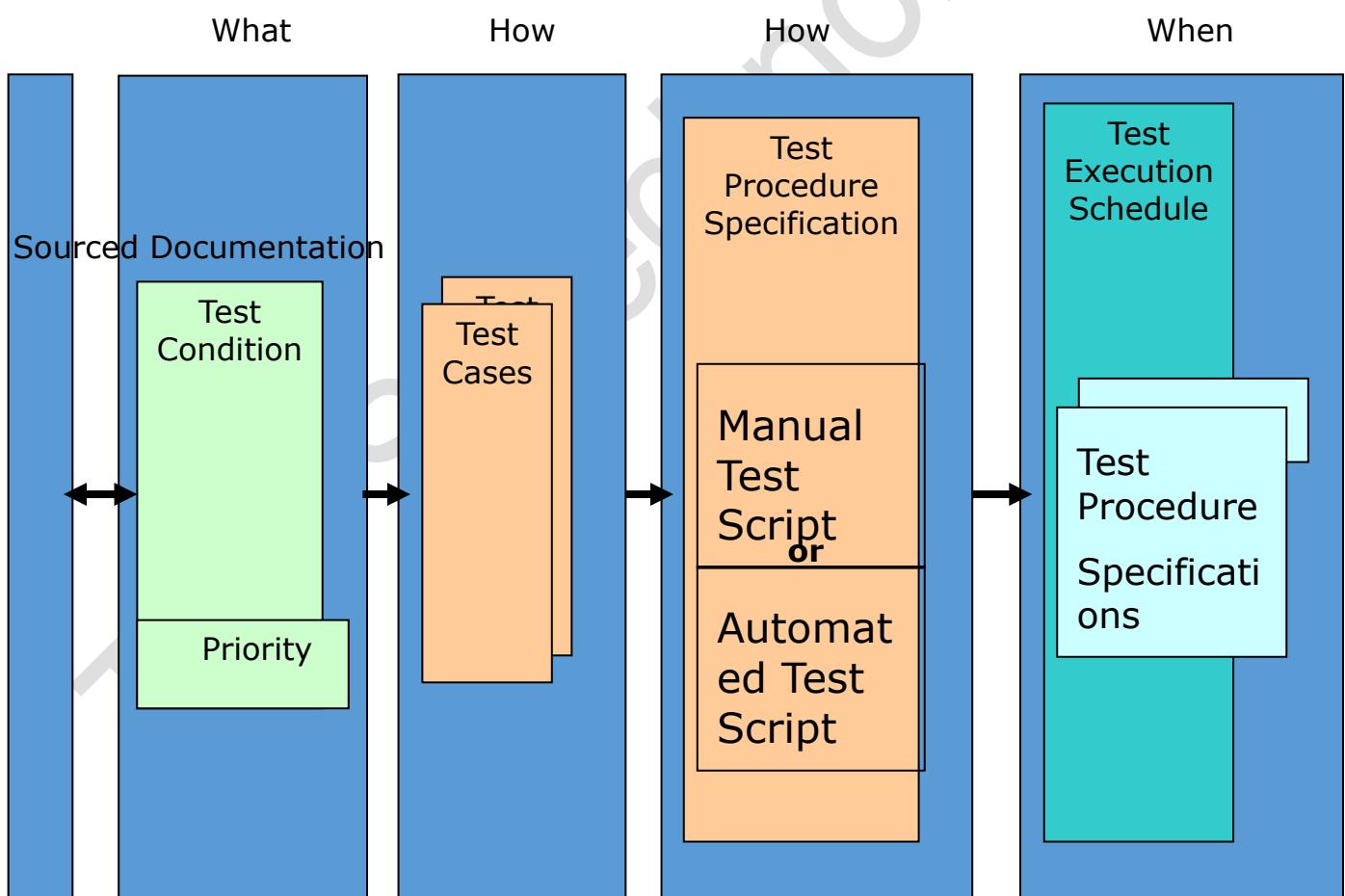
## Test Procedures Specification (Test Script)

- 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

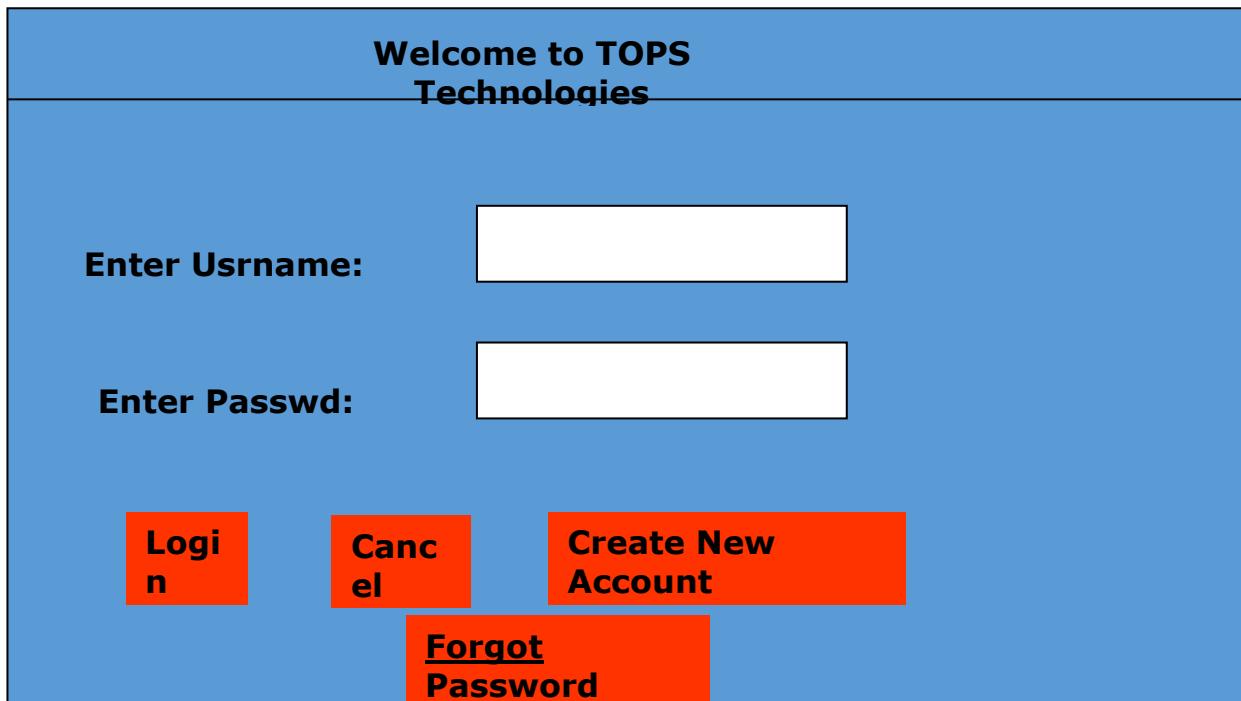
# Test Execution Schedule

- 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
  - Prioritization
  - And technical and logical dependencies

# Process of Creating Test Cases



# How to Create a Test Case



## 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).

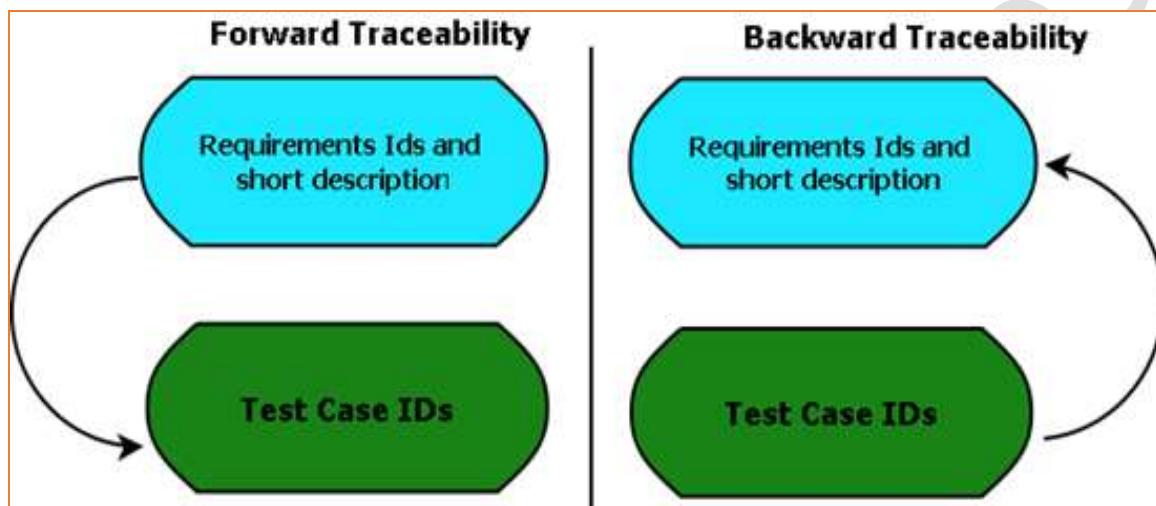
# Tractability Matrix

- 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)  
  - Traceability Matrix (also known as Requirement Traceability Matrix - RTM) is a table which is used to trace the requirements during the Software development life Cycle. It can be used for forward tracing (i.e. from Requirements to Design or Coding) or backward (i.e. from Coding to Requirements). There are many user defined templates for RTM.

- Each requirement in the RTM document is linked with its associated test case, so that testing can be done as per the mentioned requirements.
  - Furthermore, Bug ID is also include and linked with its associated requirements and test case. The main goals for this matrix are:
    - Make sure Software is developed as per the mentioned requirements.
    - Helps in finding the root cause of any bug.
  - Helps in tracing the developed documents during different phases of SDLC.
  - A requirements traceability matrix is a document that traces and maps user requirements [requirement Ids from requirement specification document] with the test case ids. Purpose is to make sure that all the requirements are covered in test cases so that while testing no functionality can be missed.

# 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

TOPS Technologies

# Example of Creating Traceability Matrix

## Steps to Create Traceability Matrix

- Make use of excel to create Traceability Matrix:
- Define following columns:
  - *Base Specification/Requirement ID (If any)*
  - *Requirement ID*
  - *Requirement description*
  - *TC 001*
  - *TC 002*
  - *TC 003... So on.*
- Identify all the testable requirements in granular level from requirement document. Typical requirements you need to capture are as follows:
  - Used cases (all the flows are captured)
  - *Error Messages*
  - *Business rules*
  - *Functional rules*
  - *SRS, FRS So on...*
- Identity all the test scenarios and test flows.
- Map Requirement IDs to the test cases. Assume (as per below table), Test case “TC 001” is your one flow/scenario. Now in this scenario, Requirements SR-1.1 and SR-1.2 are covered. So mark “x” for these requirements.
  - *Now from below table you can conclude –*
  - *Requirement SR-1.1 is covered in TC 001*
  - *Requirement SR-1.2 is covered in TC 001*
  - *Requirement SR-1.5 is covered in TC 001, TC 003 [Now it is easy to identify, which test cases need to be updated if there is any change request].*
  - *TC 001 Covers SR-1.1, SR, 1.2 [we can easily identify that test cases covers which requirements].*
  - *TC 002 covers SR-1.3.. So on..*
- This is a very basic traceability matrix format. You can add more following columns and make it more effective:
  - ID, Assoc ID, Technical Assumption(s) and/or Customer Need(s), Functional Requirement, Status, Architectural/Design Document, Technical Specification, System Component(s), Software Module(s), Test Case Number, Tested In, Implemented In, Verification, Additional Comments,

# Example of Traceability Matrix

Requirement ID	Requirement description	TC 001	TC 002	TC 003
<b>SR-1.1</b>	User should be able to do this	x		
<b>SR-1.2</b>	User should be able to do that	x		
<b>SR-1.3</b>	On clicking this, following message should appear		x	
<b>SR-1.4</b>			x	
<b>SR-1.5</b>		x		x
<b>SR-1.6</b>				x
<b>SR-1.7</b>			x	

## Fundamental Test Process

## Software Testing Life Cycle (STLC)

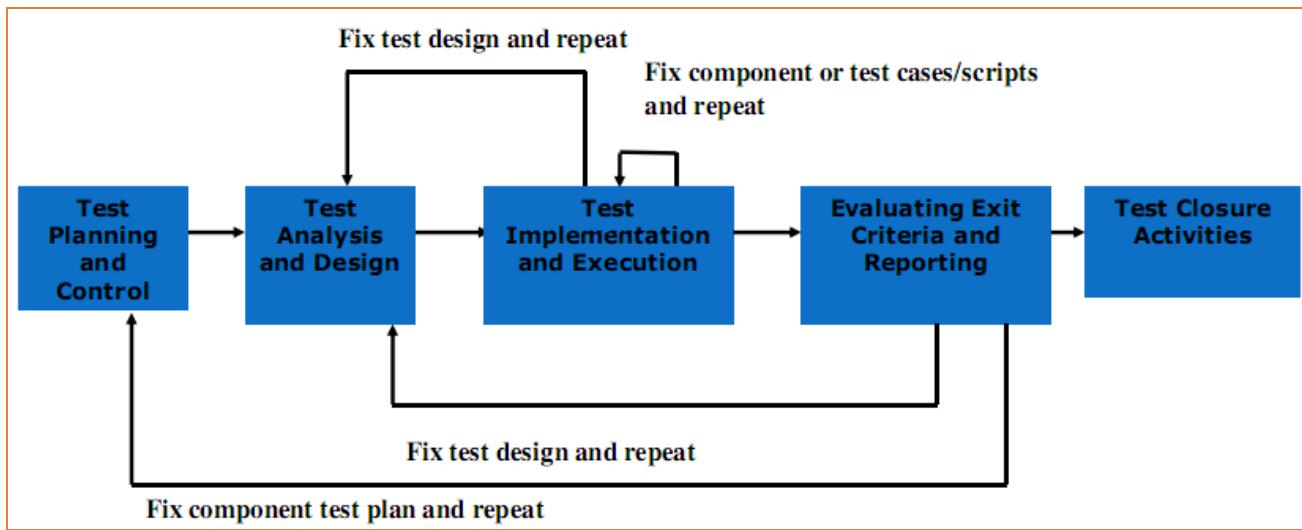
# Stages of STLC

- Test Planning and Controlling
- Test Analysis and Design
- Test Implementation and Execution
- Evaluating Exit Criteria and Reporting
- 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

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**.
- 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, Test ware, 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.
- 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 is several level of independence in software testing which is 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.
- **Clear and courteous communication and feedback on defects between tester and developer**
  - We all make mistakes and we sometimes get annoyed and upset or depressed when someone points them out.
  - So, when as testers we run a test which is a good test from our viewpoint because we found the defects and failures in the software.
  - But at the same time we need to be very careful as how we react or report the defects and failures to the programmers.

- We are pleased because we found a good bug but how will the requirement analyst, the designer, developer, project manager and customer react.
  - The people who build the application may react defensively and take this reported defect as personal criticism.
  - The project manager may be annoyed with everyone for holding up the project.
  - The customer may lose confidence in the product because he can see defects.
- Because testing can be seen as destructive activity we need to take care while reporting our defects and failures as objectively and politely as possible.

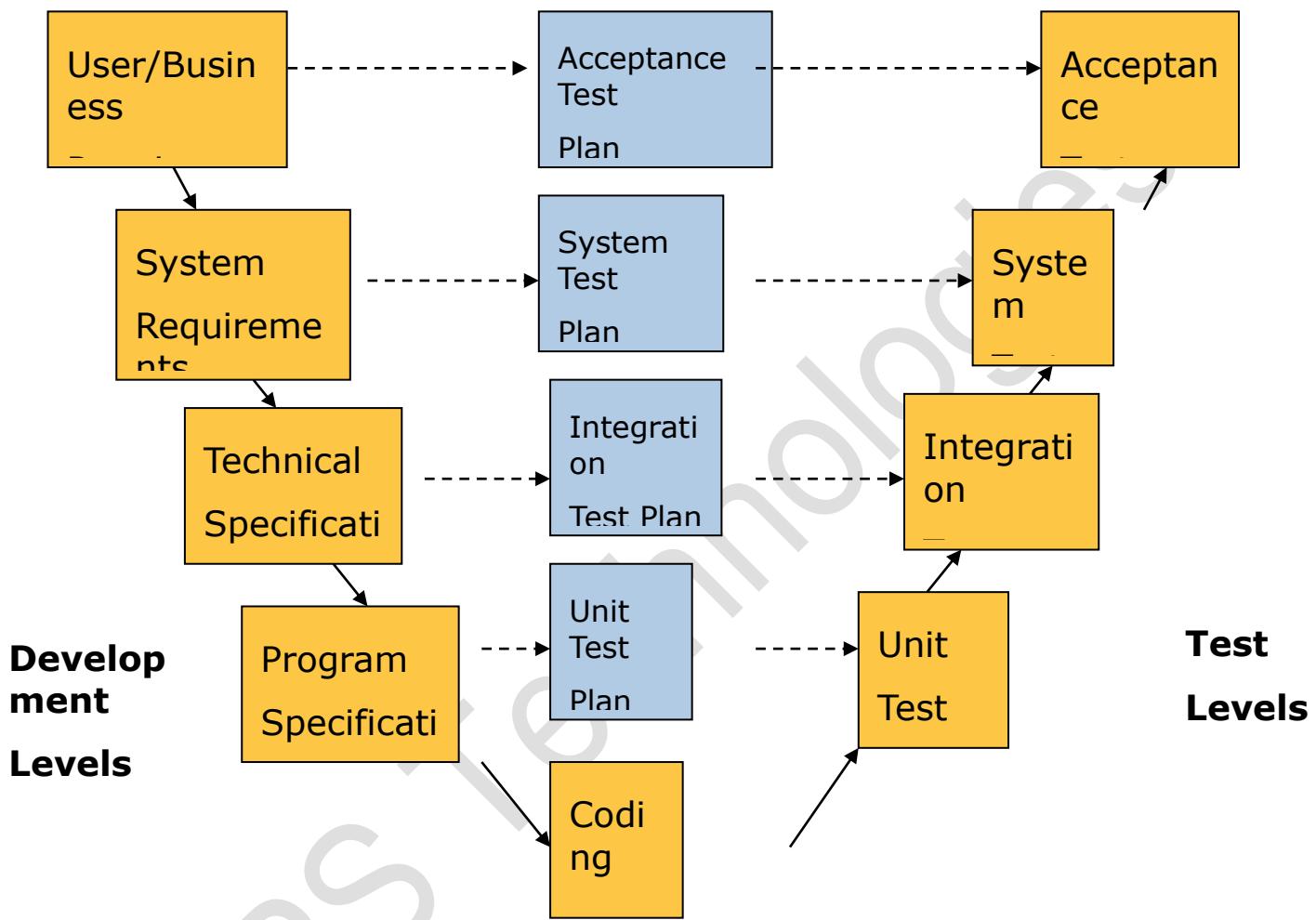
## Software Development Model

Testing does not exist in isolation; test activities are related to software development activities. Different development life cycle models need different approaches to testing.

### 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**

TOPS Technologies

# 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.
- **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.

## **Benefits of V-Model**

- The testing phases are given the same level of management attention and commitment as the corresponding development phases
- The outputs from the development phases are reviewed by the testing team to ensure their testability
- Verification and validation (and early test design) can be carried out during the development of the software work products
- The early planning and preliminary design of tests provides additional review comments on the outputs from the development phase
- The levels of development and testing shown in the model vary from project to project
  - For example, there may additional test levels, such as System Integration Testing, sitting between System Testing and Acceptance Testing (more on these test levels later)
- The work products coming out from any one development level may be utilised in one or more test levels
  - For example, whilst the prime source for Acceptance testing is the Business Requirement, the System Requirements (e.g. Use Cases) may also be needed to support detailed test design

# Verification & Validation Phase

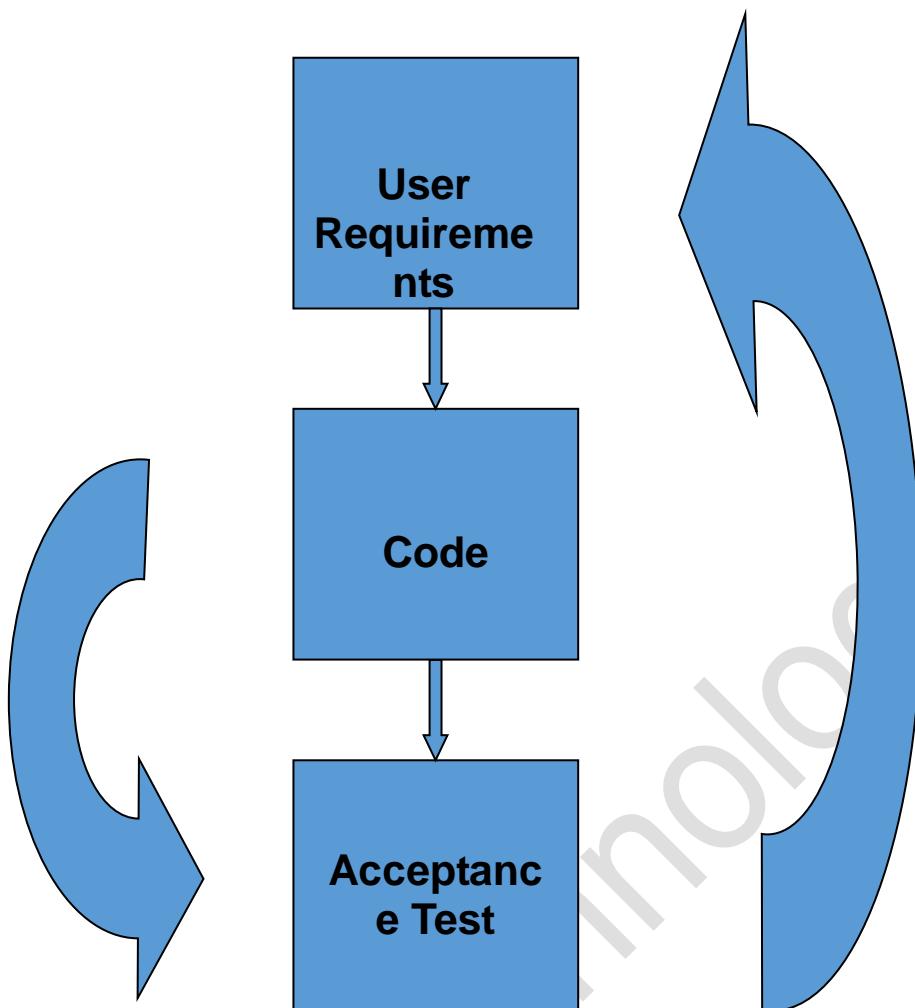
Criteria	Verification	Validation
<b>Definition</b>	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
<b>Objective</b>	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.
<b>Question</b>	Are we building the product right?	Are we building the right product?
<b>Evaluation Items</b>	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
<b>Activities</b>	<ul style="list-style-type: none"> <li>• Reviews</li> <li>• Walkthroughs</li> <li>• Inspections</li> </ul>	<ul style="list-style-type: none"> <li>• Testing</li> </ul>

# Iterative Development Models

- Iterative Development
  - Establish Requirements
  - Design the System
  - Build the System
  - Test the System
- Achieved with small developments – Iterations and Increment within Iterations
- As Increments are developed and tested the System grows and grows. Need for more testing with Regression Testing paramount
- E.g. RAD, RUP and Agile development models
- Agile development
  - Aim is to deliver software early and often
  - Rapid production and ‘time to market’
  - Can handle (and anticipates) changing requirements throughout all development and test phases

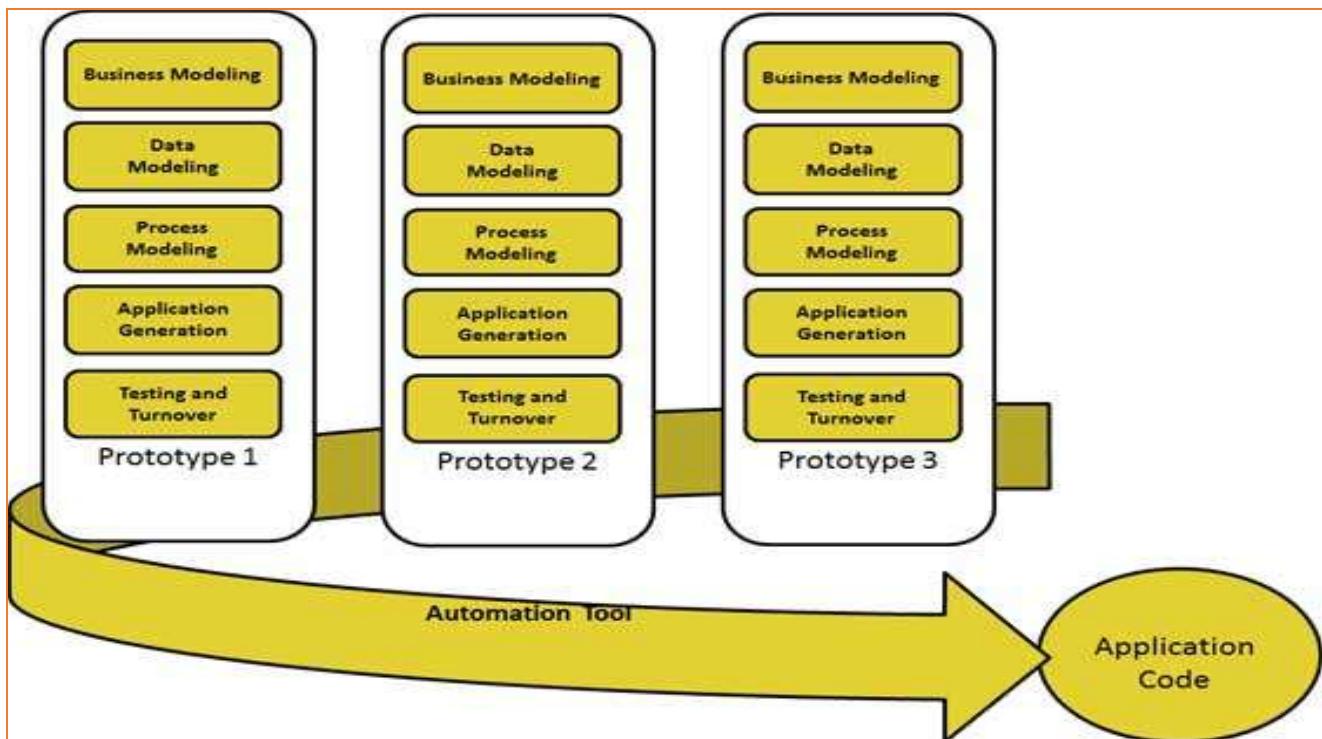
## RAD Model

- 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.



- 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 vs Traditional SDLC

- The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.
- The customer may need some changes after he actually gets to see the software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.
- RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

## 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.

# Testing within a Life Cycle Model

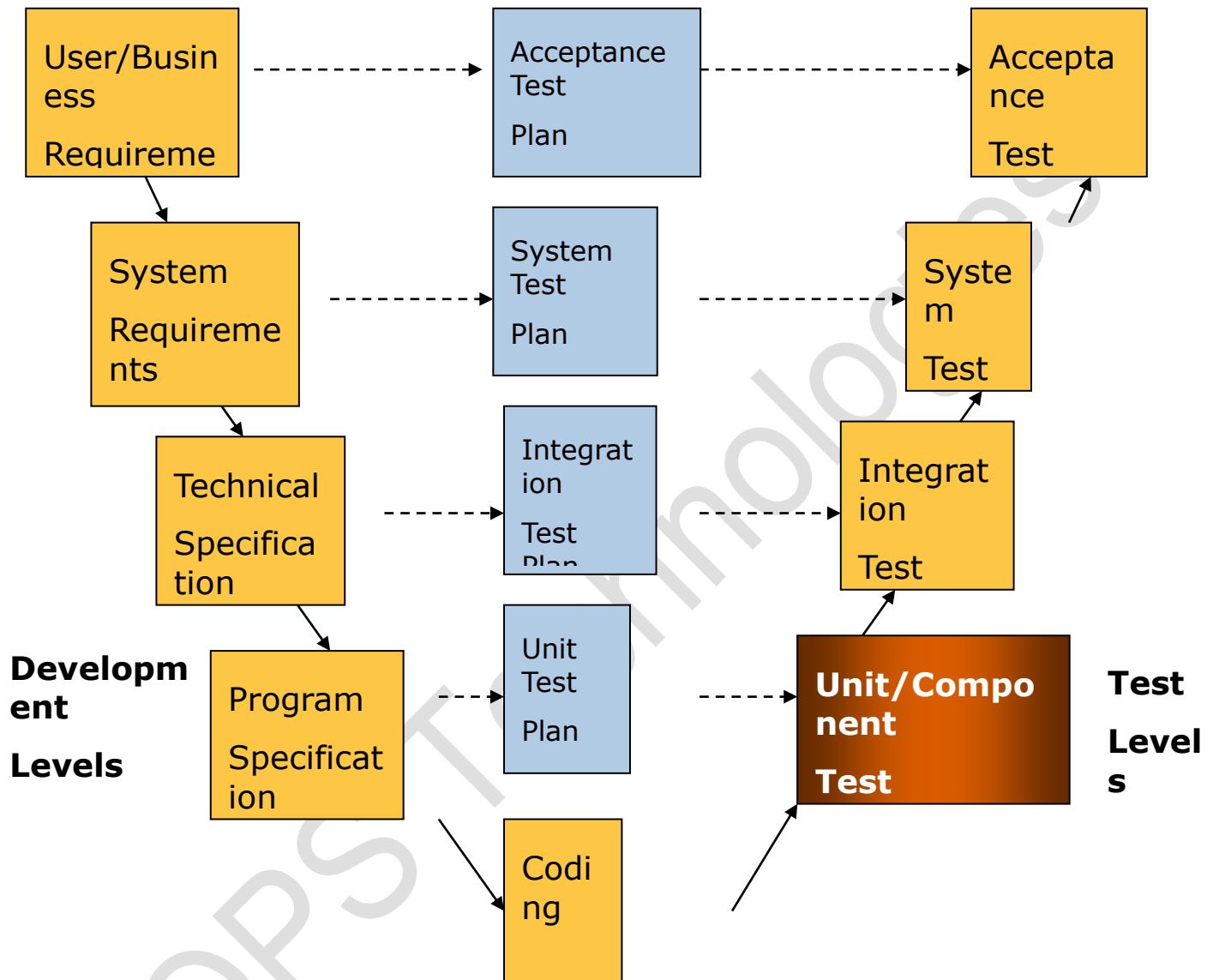
- In any life cycle model, there are several characteristics of good testing:
  - For every development activity there is a corresponding testing activity.
  - Each test level has test objectives specific to that level.
  - The analysis and design of tests for a given test level should begin during the corresponding development activity.
  - Testers should be involved in reviewing documents as soon as drafts are available in the development life cycle.
- Test levels can be combined or reorganized depending on the nature of the project or the system architecture. For example, for the integration of a commercial off-the-shelf (COTS) software product into a system, the purchaser may perform integration testing at the system level (e.g. integration to the infrastructure and other systems, or system deployment) and acceptance testing (functional and/or nonfunctional, and user and/or operational testing).

## Software Testing Levels

## Agenda

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

# Component Testing



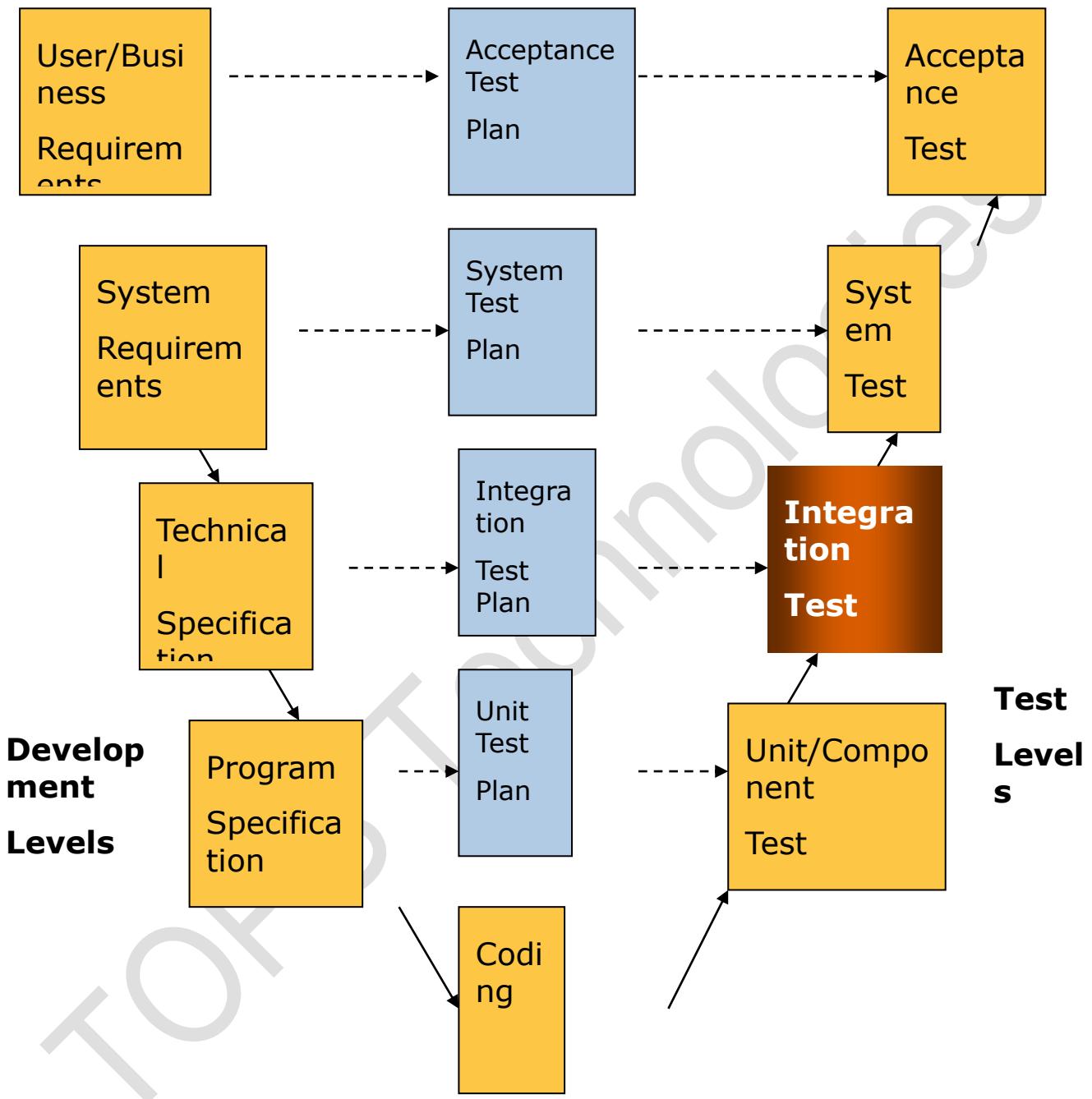
# Component Testing

- **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 tool**.
- 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.**
- 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:
  - Design test that defines how you think a small part of the software should behave (Incremental development).
  - 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!
  - 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
- **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 Integration Testing:
  - Big 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

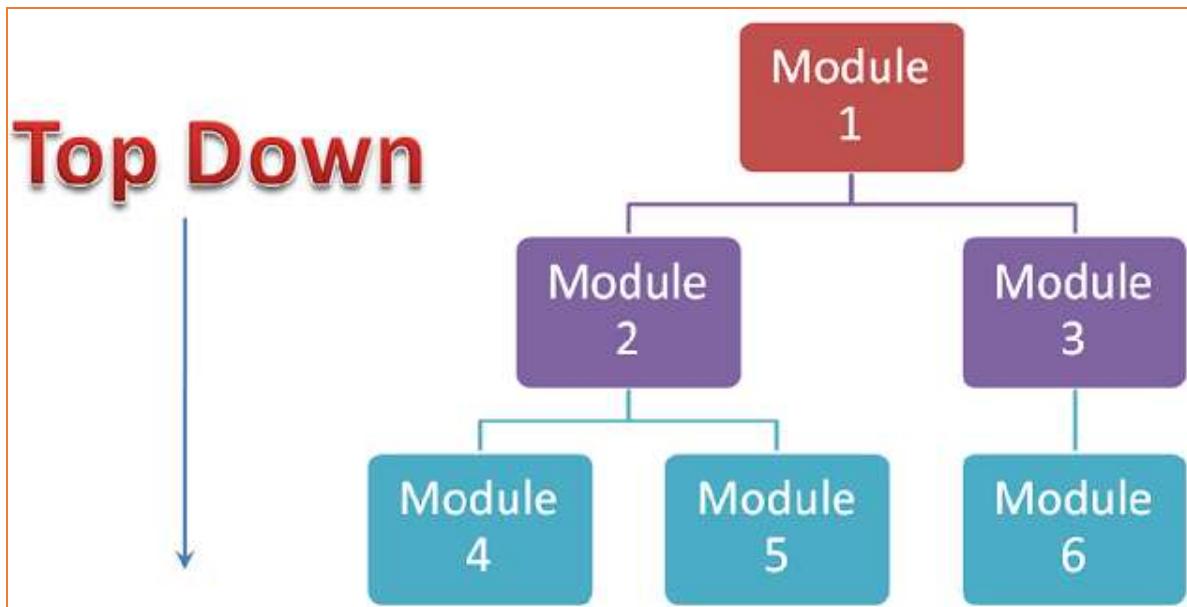
- **Advantages:**
  - Convenient for small systems.
- **Disadvantages:**
  - Fault Localization is difficult.
  - Given the sheer number of interfaces that need to be tested in this approach, some interfaces links to be tested could be missed easily.
  - Since the integration testing can commence only 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

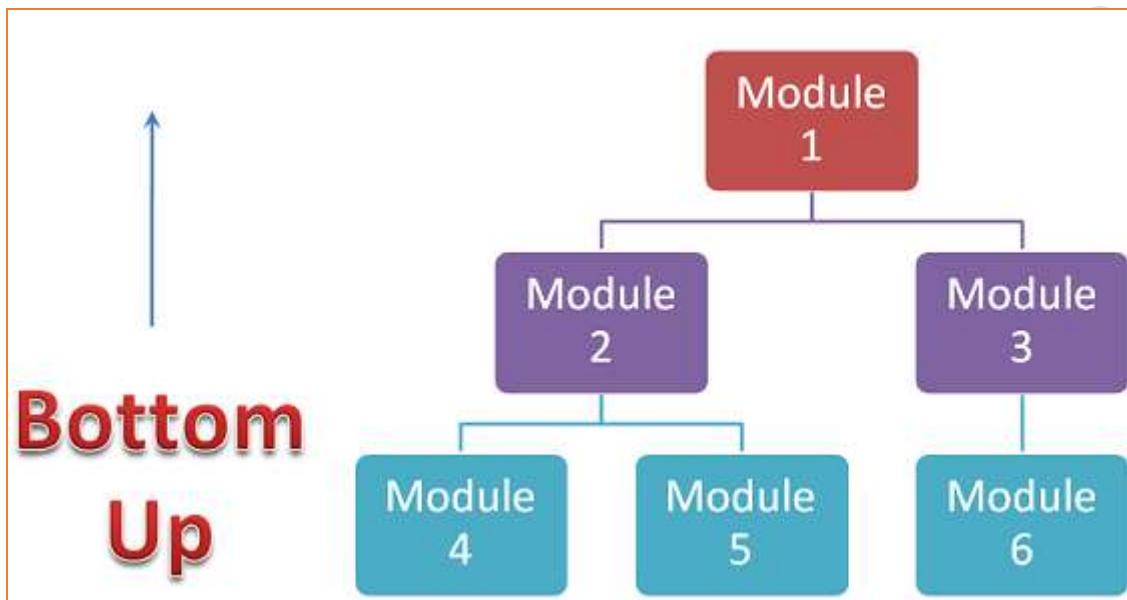
- Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs.
- In Top to down approach, testing takes place from top to down following the control flow of the software system.
- Takes help of stubs for testing.



- **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



- **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

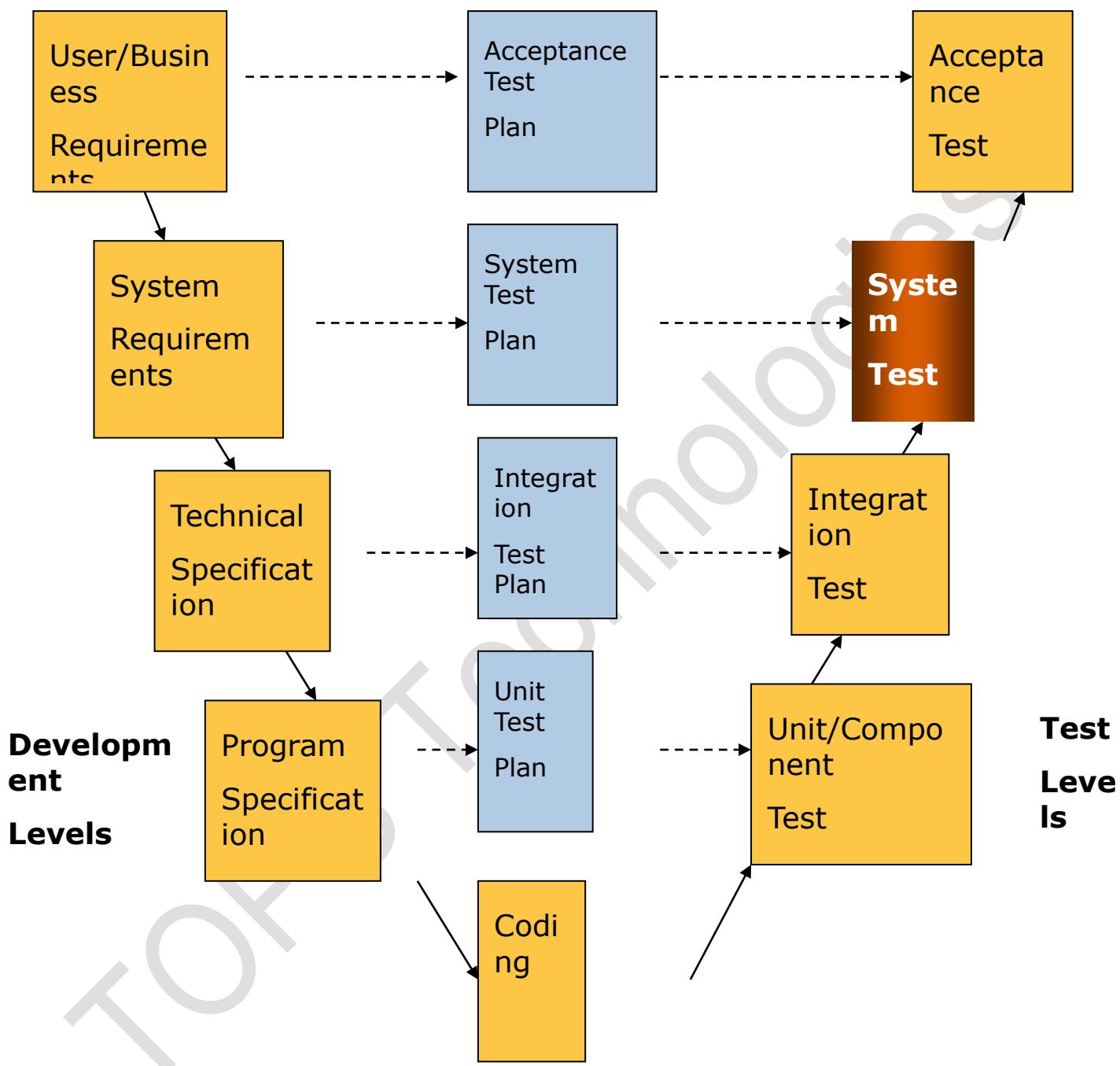
# Entry and Exit Criteria

- **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.

TOPS Technologies

# 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**
- **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:**

<b>Accuracy</b>	Provision of right or agreed results or effects
<b>Interoperability</b>	Ability to interact with specified systems
<b>Compliance</b>	Adhere to applicable standards, conventions, regulations or laws
<b>Auditability</b>	Ability to provide adequate and accurate audit data
<b>Suitability</b>	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.

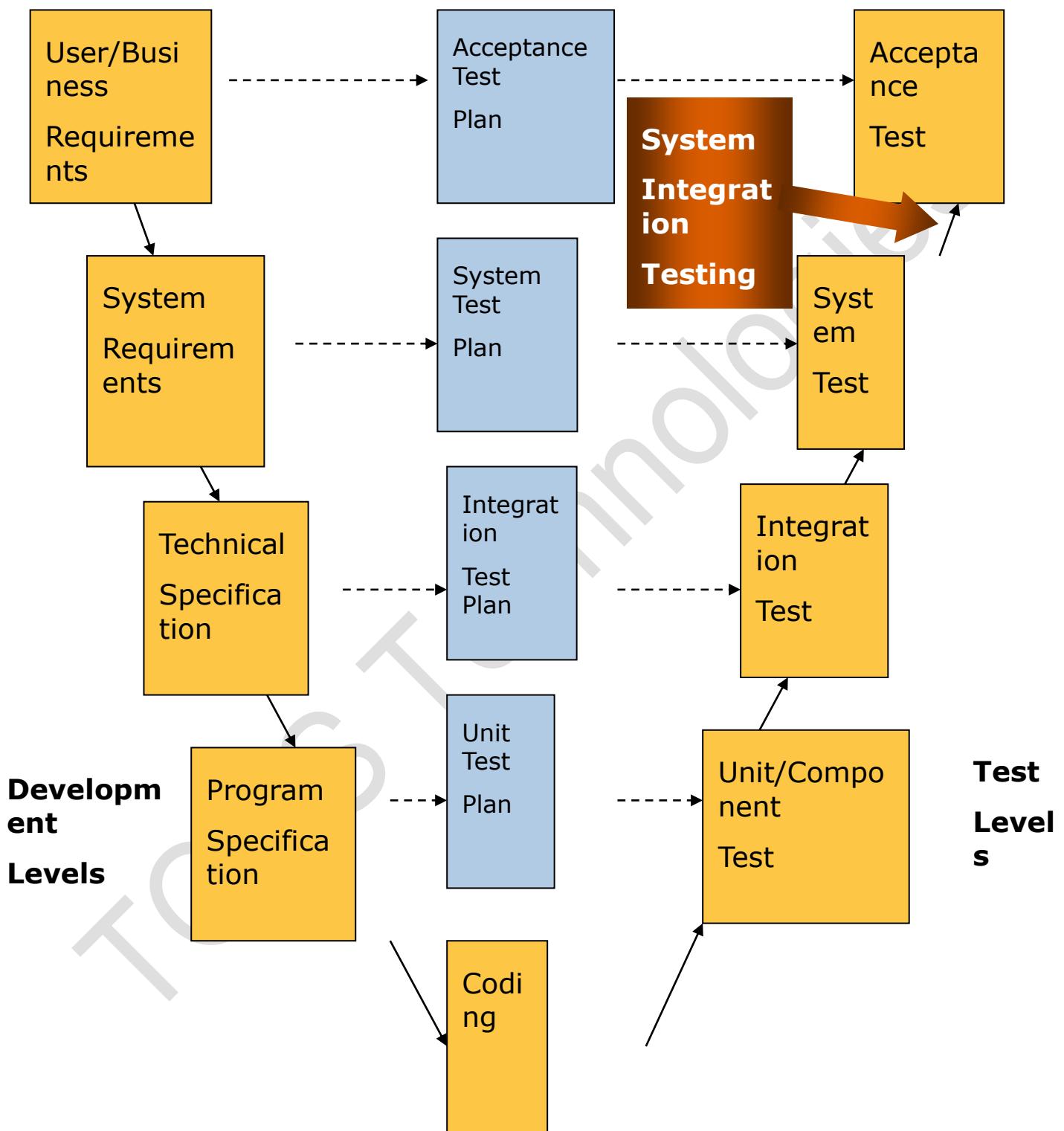
# Non-functional System Testing

- **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.**’
- 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**

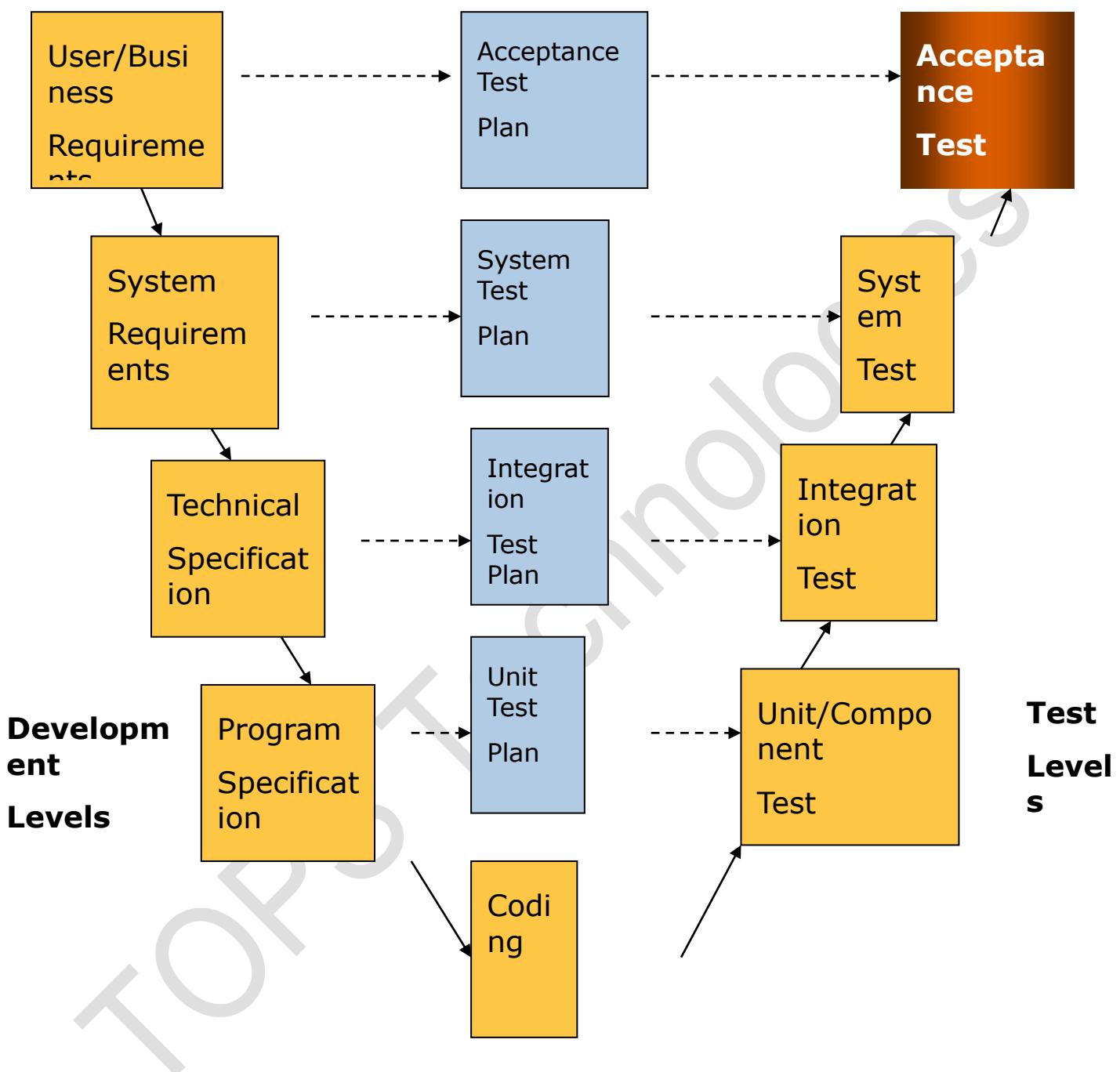
# System Integration Testing



# 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 organizations.
- 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 effect on other systems that may also be present in the **live environment**, or vice versa
- 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**.
- 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.
- **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.
- 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 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.

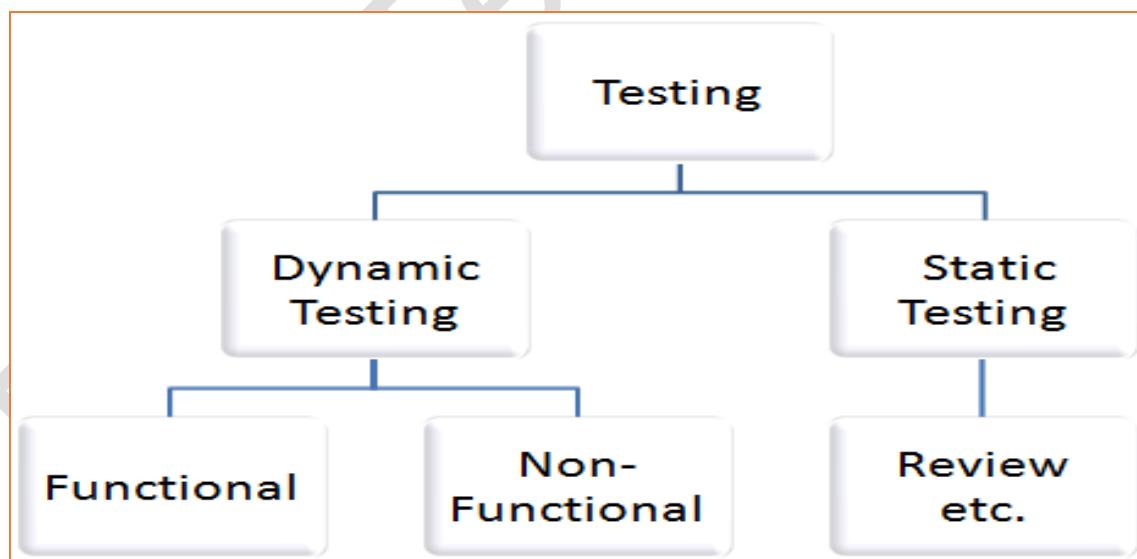
# Testing Definitions as per ISTQB

- **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

- A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of defect and relatively poor at finding other types.
- For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs. Similarly, testing performed at different stages in the software development life cycle will find different types of defects; component testing is more likely to find coding logic defects than system design defects.
- **Dynamic technique**
  - Specification-based (black-box, also known as behavioral techniques)
  - Structure-based (white-box or structural techniques)
  - Experience- based
- **Static technique**



# Test Types – The Targets of Testing

- **Target For Testing** - A group of test activities aimed at verifying the software system (or a part of a system) based on a specific reason.
- **Test type** - A group of test activities aimed at testing a component or system regarding one or more interrelated quality attributes. A test type is focused on a specific test objective, e.g.
  - reliability test
  - usability test
  - Structure or architecture of the system/software
  - Regression test and may take place on one or more test levels or test phases.
  - A model of the software may be developed and/or used in structural and functional testing For example
- In Functional Testing
  - a process flow model
  - a state transition model
  - a plain language specification
- For Structural Testing
  - a control flow model
  - a menu structure model

## 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

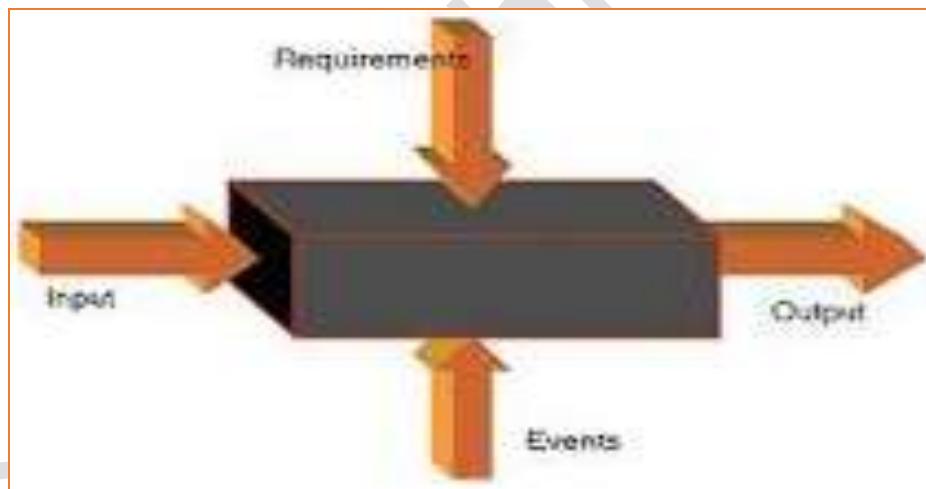
- Under **Static Testing** code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Hence, the name “static”.
- Main objective of this testing is to improve the quality of software products by finding errors in early stages of the development 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
- 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
- **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.
- **Unit Testing:** Under unit testing, individual units or modules is 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 between 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.
Requires loads of meetings	Comparatively requires lesser meetings

# Dynamic Testing Techniques

## 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

# 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.
- 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**

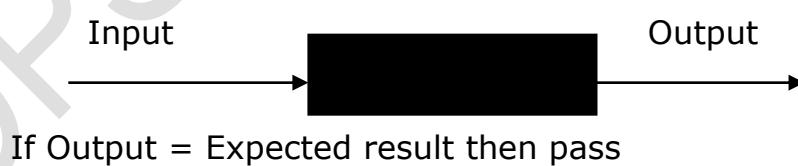
# Functional vs Non-Functional

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"> <li>• Unit Testing</li> <li>• Smoke Testing</li> <li>• Sanity Testing</li> <li>• Integration Testing</li> <li>• White box testing</li> <li>• Black Box testing</li> <li>• User Acceptance testing</li> <li>• Regression Testing</li> </ul>	Types of Nonfunctional testing are <ul style="list-style-type: none"> <li>• Performance Testing</li> <li>• Load Testing</li> <li>• Volume Testing</li> <li>• Stress Testing</li> <li>• Security Testing</li> <li>• Installation Testing</li> <li>• Penetration Testing</li> <li>• Compatibility Testing</li> <li>• Migration Testing</li> </ul>

# Black Box Testing and Techniques

## 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.
  - 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**.



- **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.

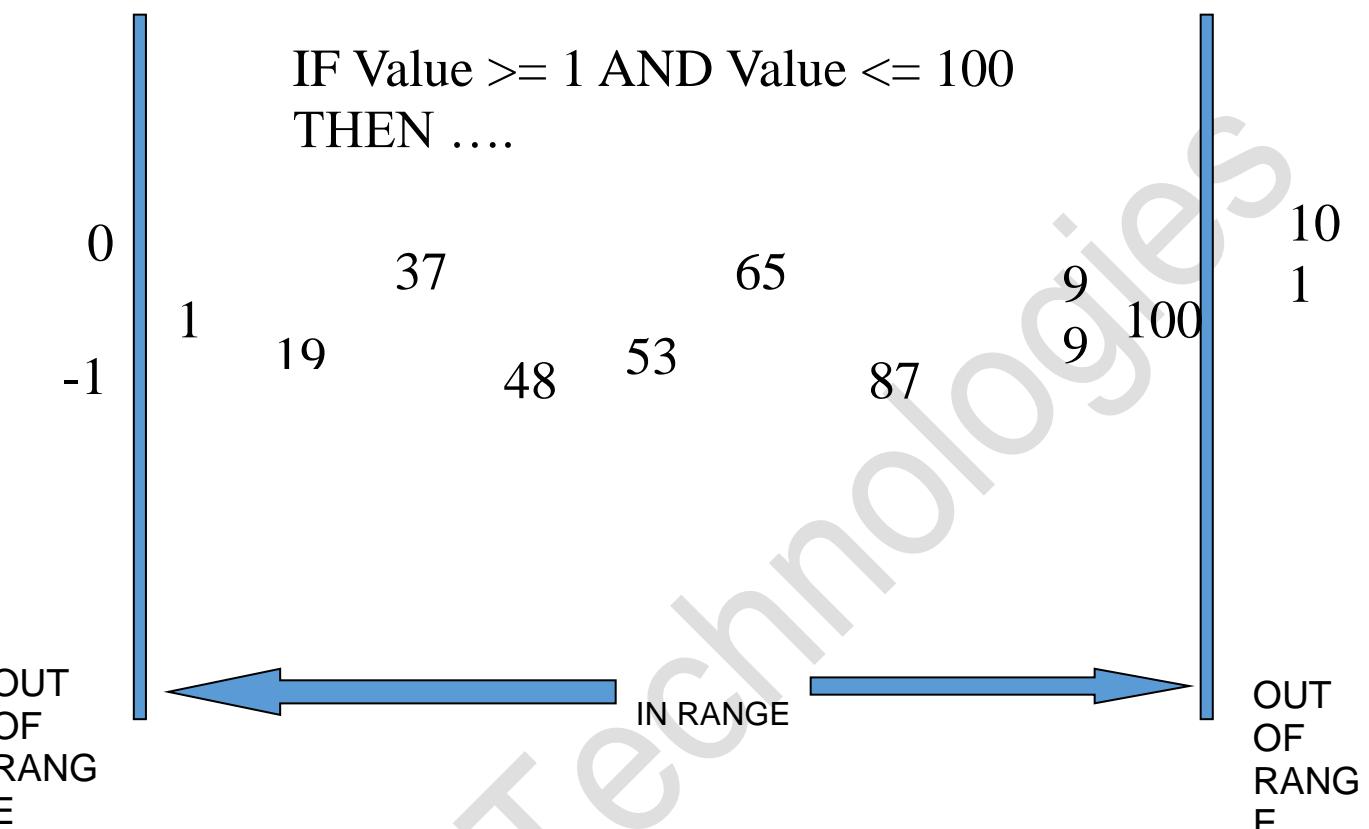
## Techniques of Black Box Testing

- 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.)

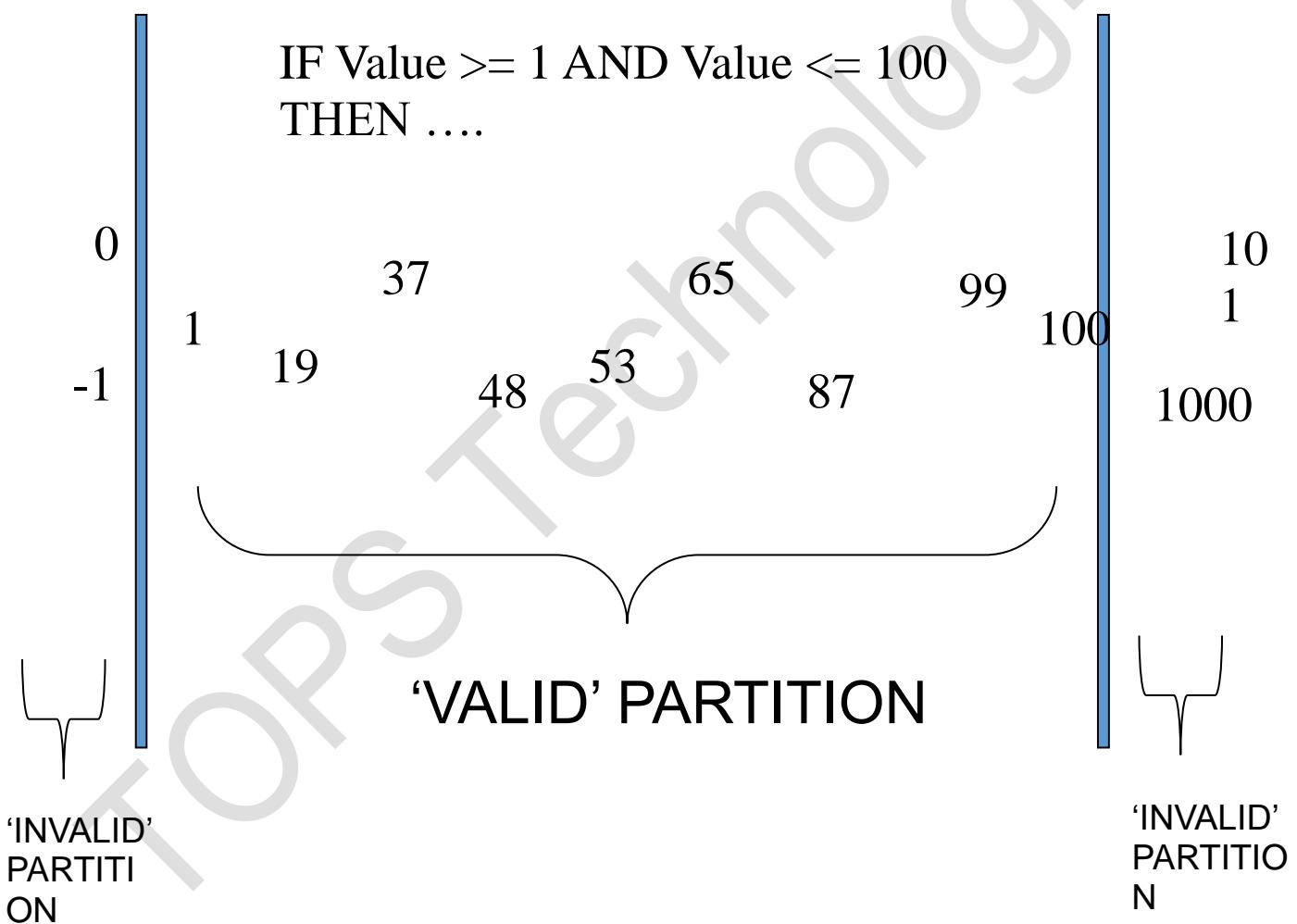
- Aim is to treat groups of inputs as equivalent and to select one representative input to test them all
- EP can be used for all Levels of Testing
- Equivalence partitioning is the process of defining the optimum number of tests by:
  - Reviewing documents such as the Functional Design Specification 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  $\geq 1$  and value  $\leq 100$ ) Then...”

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



# 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

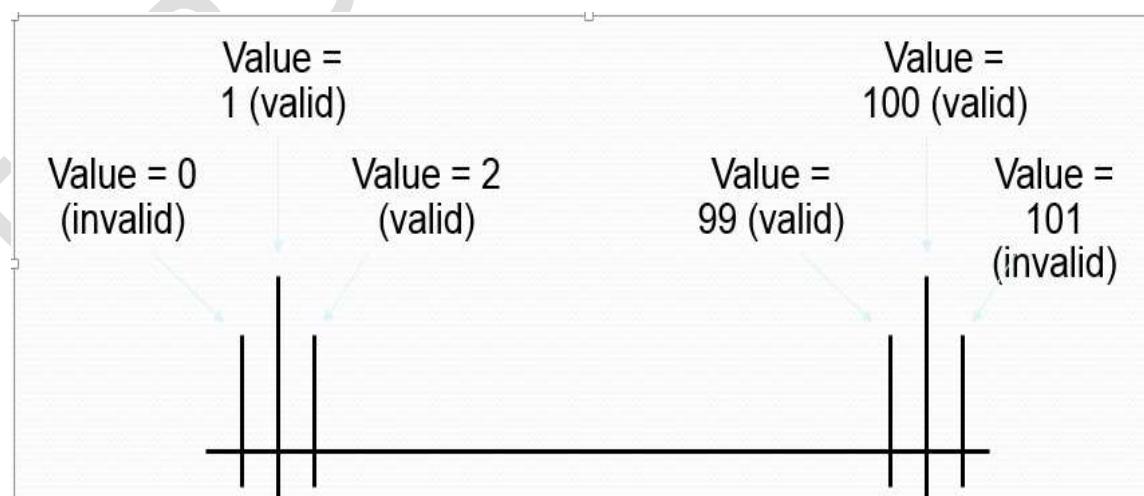


# Equivalence Partitioning (E.P.)

- 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.)

- 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].
- 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	F
Input 2	T	T	F
Input 3	T	DON'T CARE	F
Input 4	T	F	T
Output/Response			
Response 1	Y	Y	N
Response 2	Y	N	Y
Response 3	N	Y	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

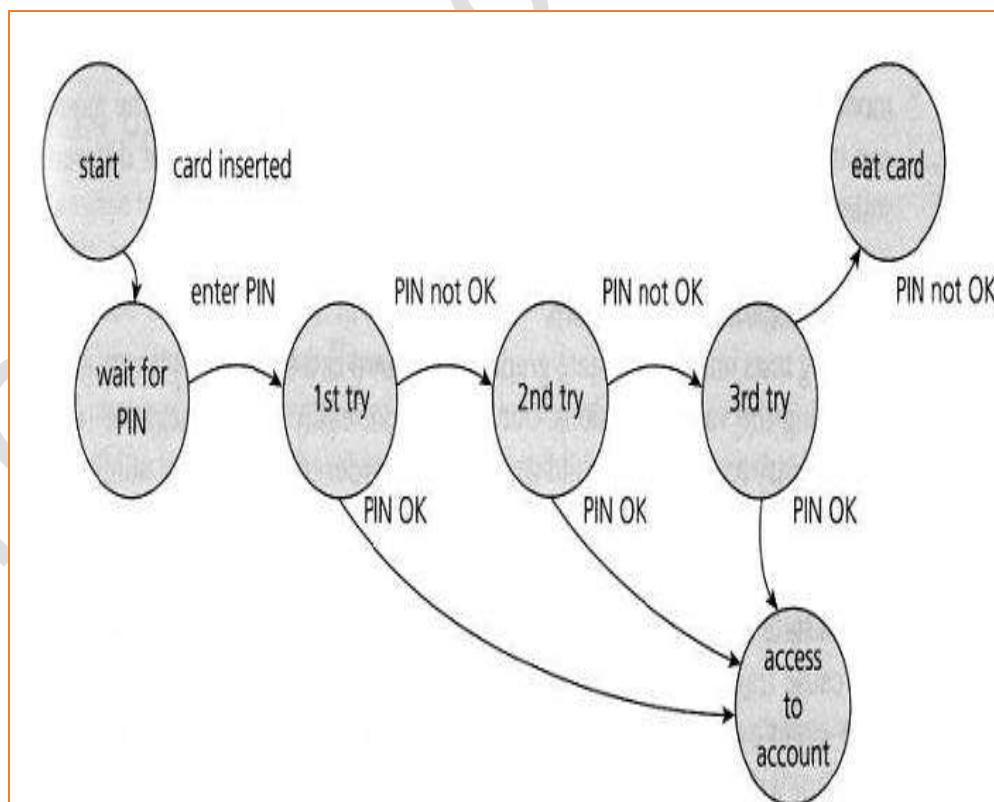
# State Transaction Testing

- 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
  - Web site transitions

# State Transaction Testing

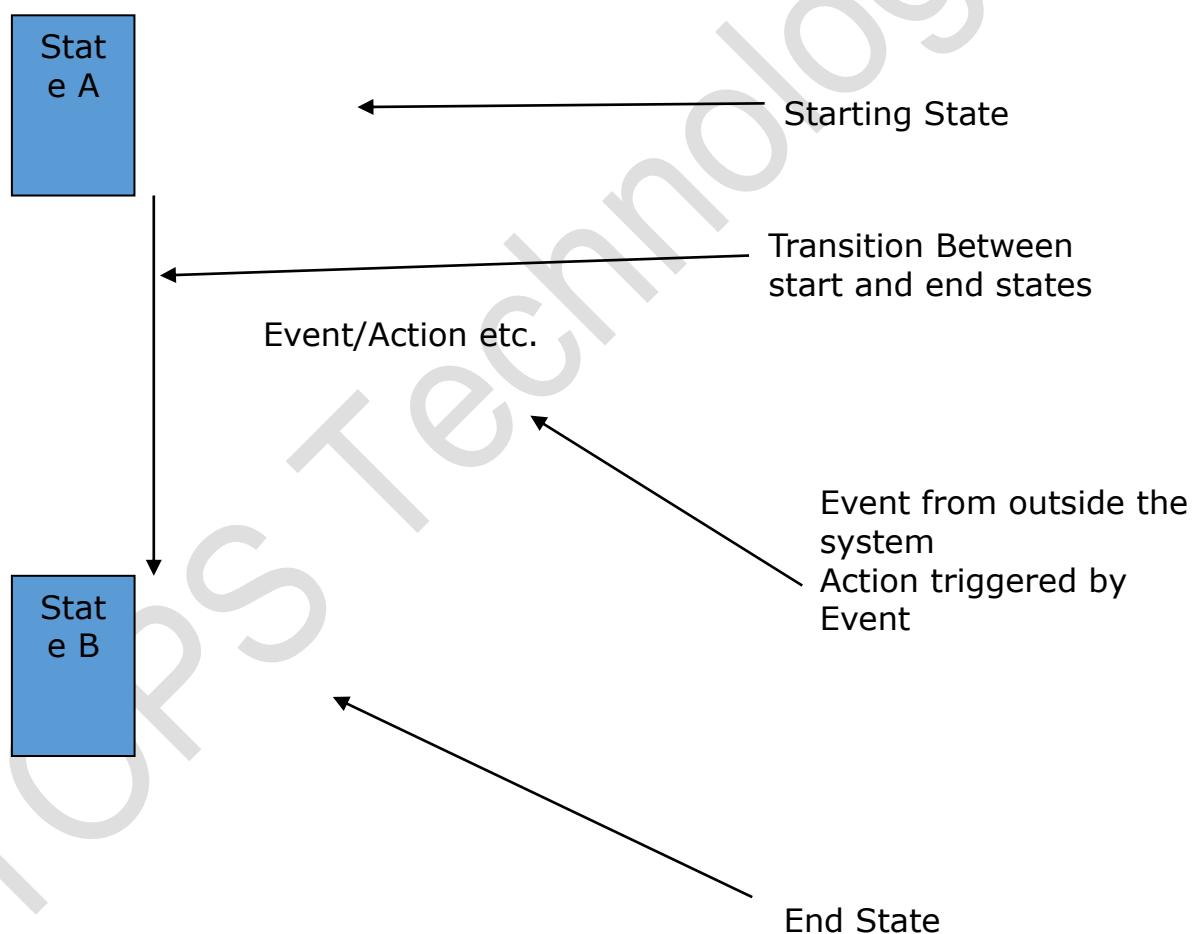
- State transition testing uses the same principles as the State Transition Diagramming design technique.
- State transition testing is used where some aspect of the system can be described in what is called a '**finite state machine**'. This simply means that the system can be in a (finite) number of 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

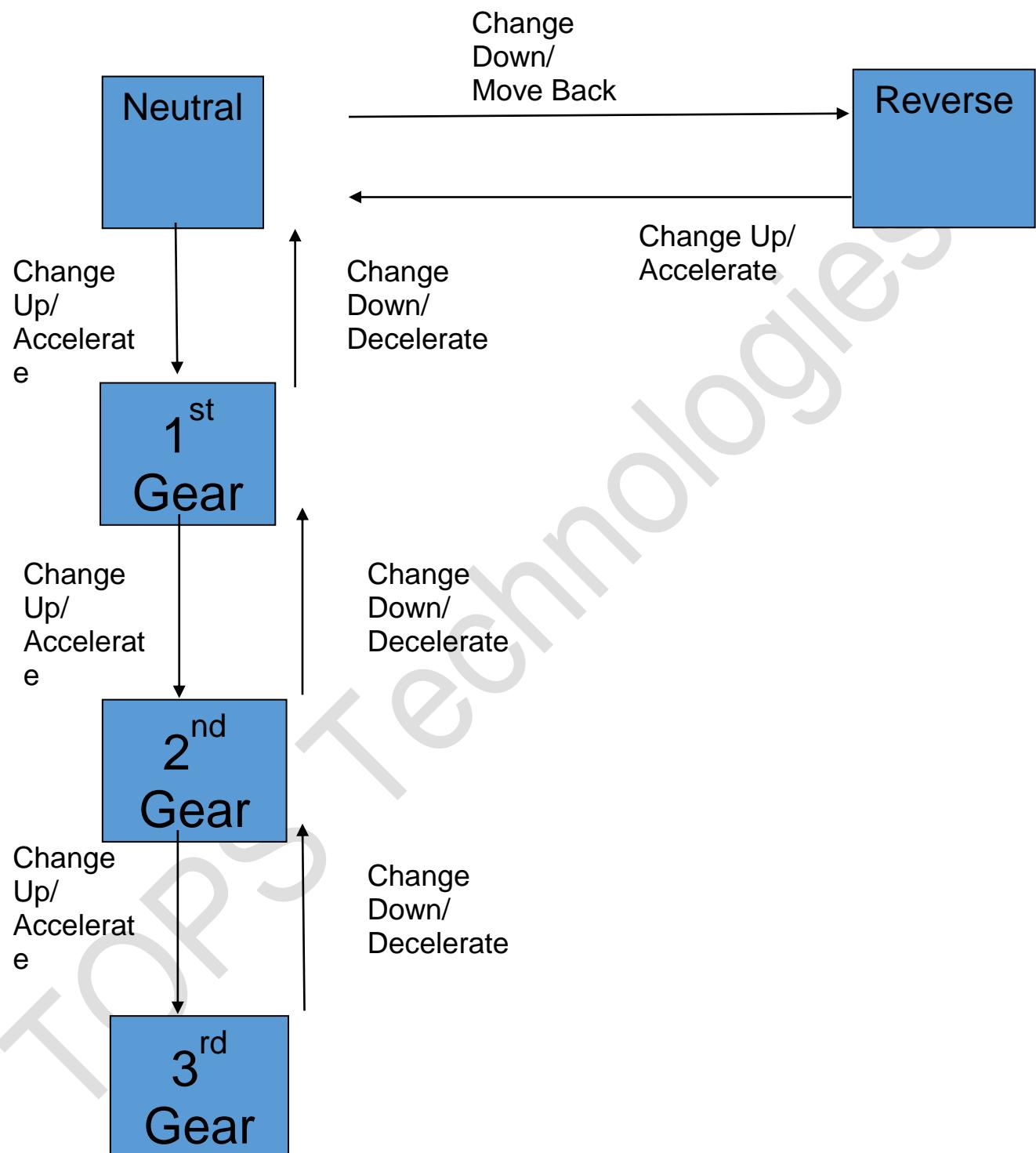


# State Transaction Example

- 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



# 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)

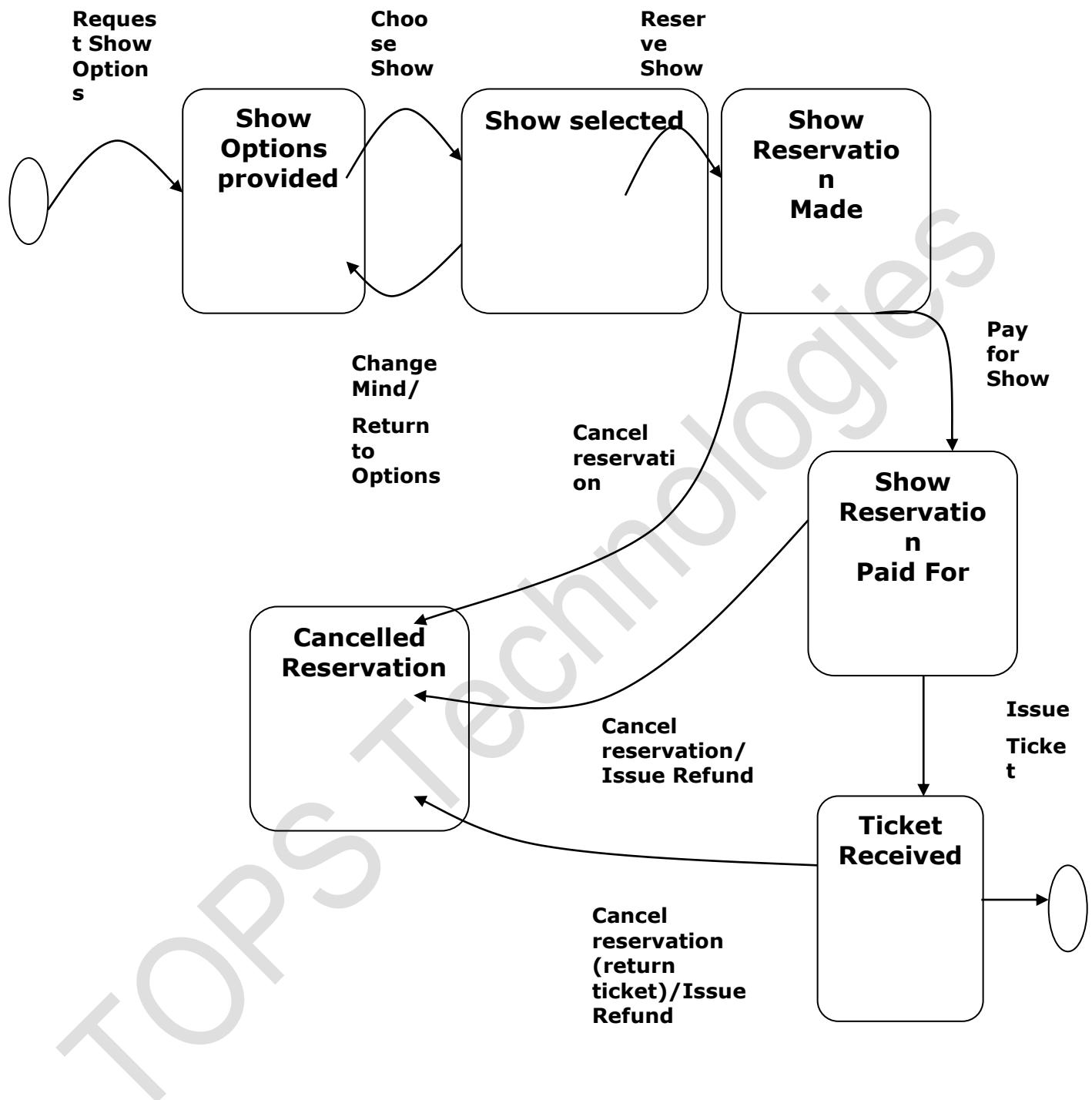
R→N→1  
R→N→R  
N→R→N  
N→1→2  
N→1→N  
1→N→R  
Etc.

# 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

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

# State Transaction Example



# State Table of State Transaction Example

Current State	Event/ Next State					
	A	B	C	D	E	F
SS	1					
1		2				
2			1	3		
3					4	
4						ES
ES						

## Use Case Testing

- Use cases are a method of describing requirements
- Structured approach to requirements design
- Usually has one main flow (though may also have alternate flows)
- Each use case is a process flow or scenario
- Particularly useful in determining tests for (Functional) Systems Test and for UAT
- Also good at detecting defects in the integration of interfaces
- The main parts of a Use Case are:
  - Actors users of the system
  - Pre-conditions What are the starting requirements for the use case
  - Post conditions The state the system will end up in once completed

# Use Case Testing Example

- The ATM PIN example is shown below in Figure. We show successful and unsuccessful scenarios.
- In this diagram we can see the interactions between the A (actor – in this case it is a human being) and S (system).
- From step 1 to step 5 that is success scenario it shows that the card and pin both got validated and allows Actor to access the account.
- But in extensions there can be three other cases that is 2a, 4a, 4b which is shown in the diagram below.
- For use case testing, we would have a test of the success scenario and one testing for each extension. In this example, we may give extension 4b a higher priority than 4a from a security point of view.

	Step	Description
<b>Main Success Scenario</b>  <b>A: Actor</b> <b>S: System</b>	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
<b>Extensions</b>	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

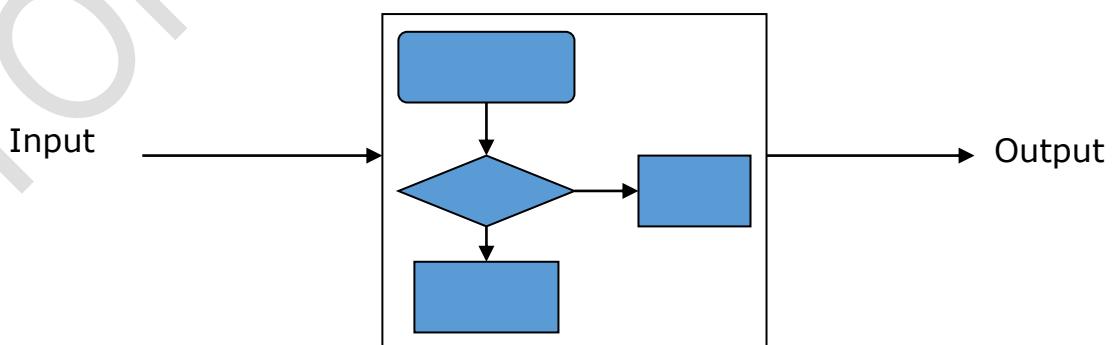
# Other Black Box Techniques

- 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

# White Box Testing and Techniques

## 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.
  - 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 ...

# 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.

## Where to Apply Test Coverage?

- Test coverage can be used in any level of the testing.
- Test coverage can be measured based on a number of different structural elements in a system or component.
- **Coverage can be measured at component testing level, integration-testing level or at system- or acceptance-testing levels.**
- **For example**, at system or acceptance level, the coverage items may be requirements, menu options, screens, or typical business transactions.
- At integration level, we could measure coverage of interfaces or specific interactions that have been tested.
- The coverage measures for specification-based techniques would apply at whichever test level the technique has been used (e.g. system or component level).

## 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:**

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

- **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** unlike 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

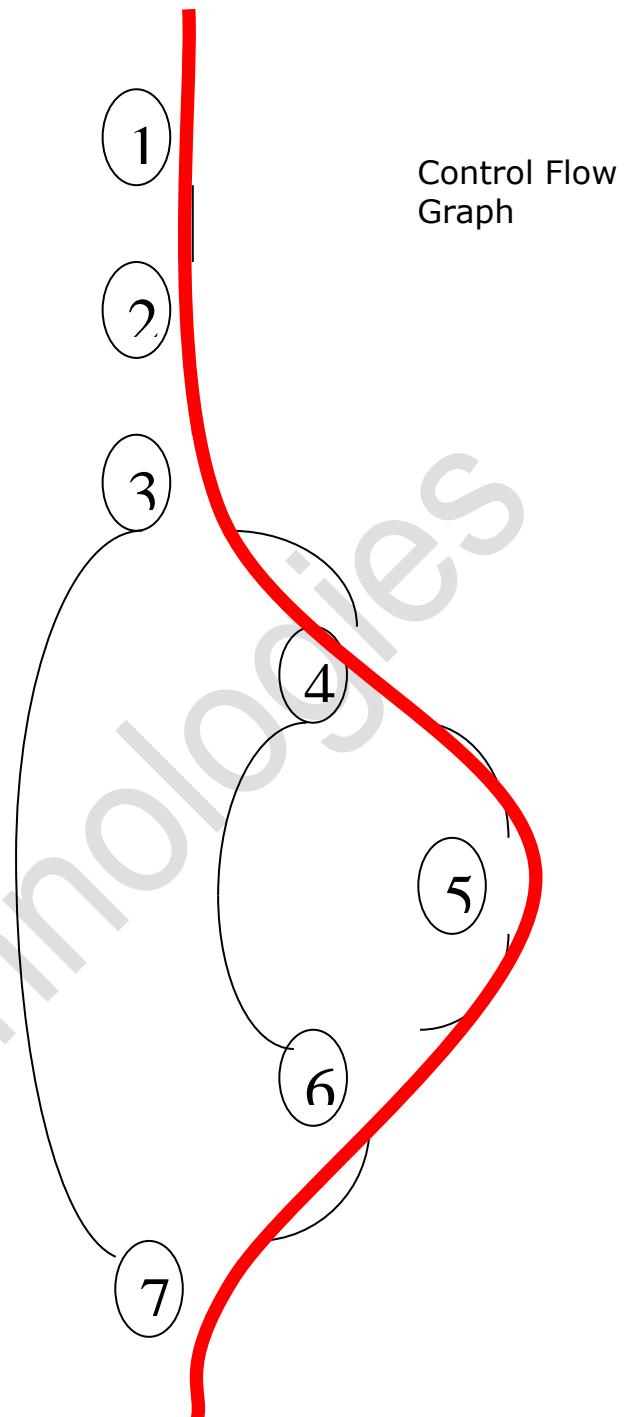
- This is closely related to decision coverage but has better sensitivity to the control flow.
- However, **full condition coverage does not guarantee full decision coverage.**
- Condition coverage reports the true or false outcome of each condition.
- Condition coverage measures the conditions independently 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

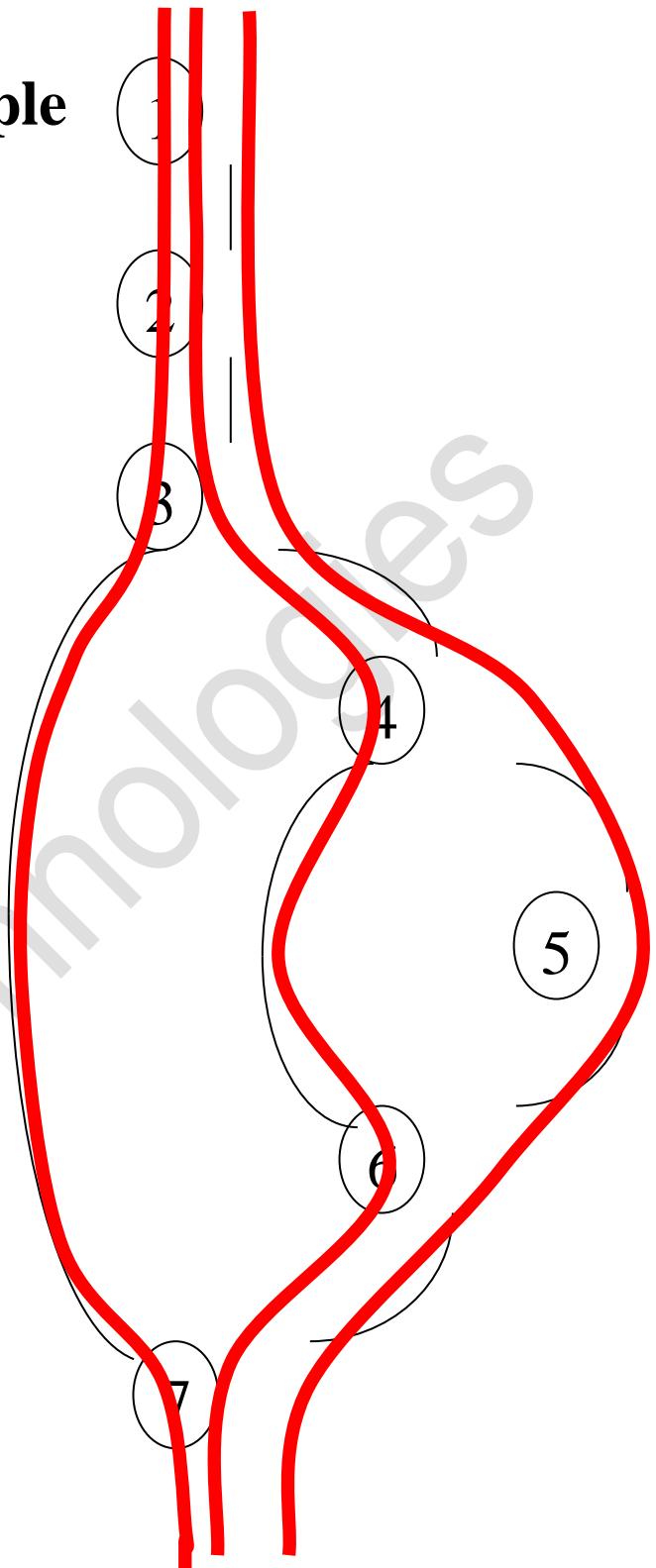
1. Read vehicle
2. Read colour
3. If vehicle = 'Car' Then
4.     If colour = 'Red' Then
5.             Print "Fast"
6.     End If
7. End If

- **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 =**



# Code Cover in Flow Chart

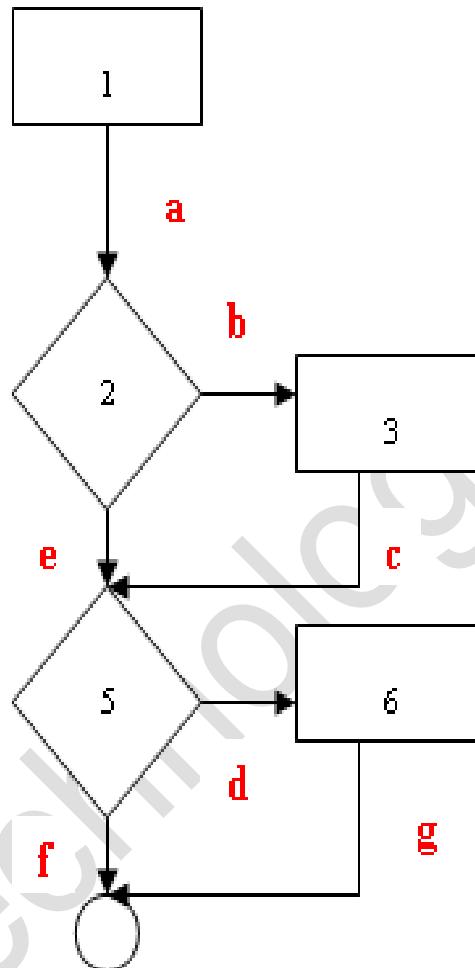
1. Read A
2. If  $A > 40$  Then
3.      $A = A * 2$
4. End If
5. If  $A > 100$  Then
6.      $A = A - 10$
7. End If

- **Answer :**

- **Decision/Branch Coverage is 2.**

- **Reason:**

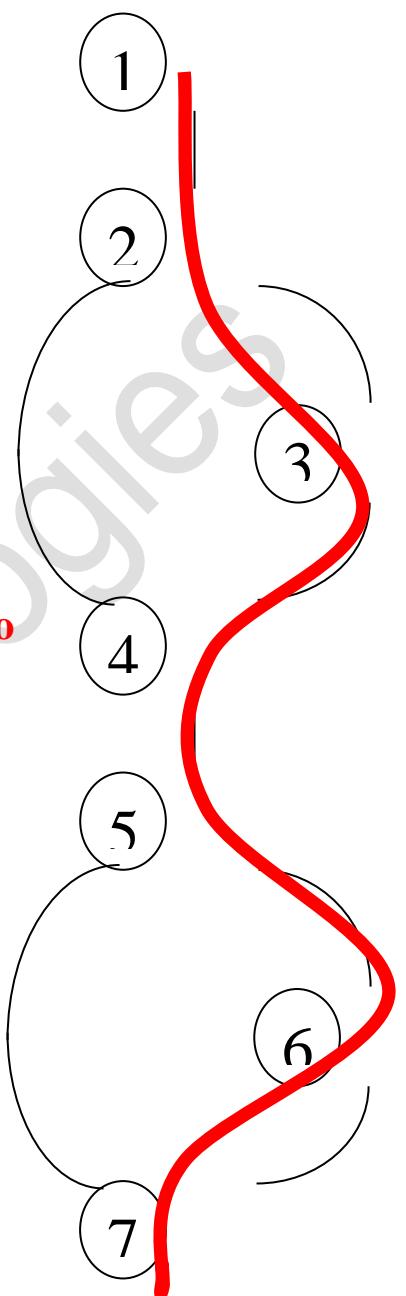
- **Because of we need to take two input to cover the four branches**
- **Consider Input is:**
  - $A = 60$  cover branch no 2 and 4
  - $A = 10$  cover branch no 1 and 3



# 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

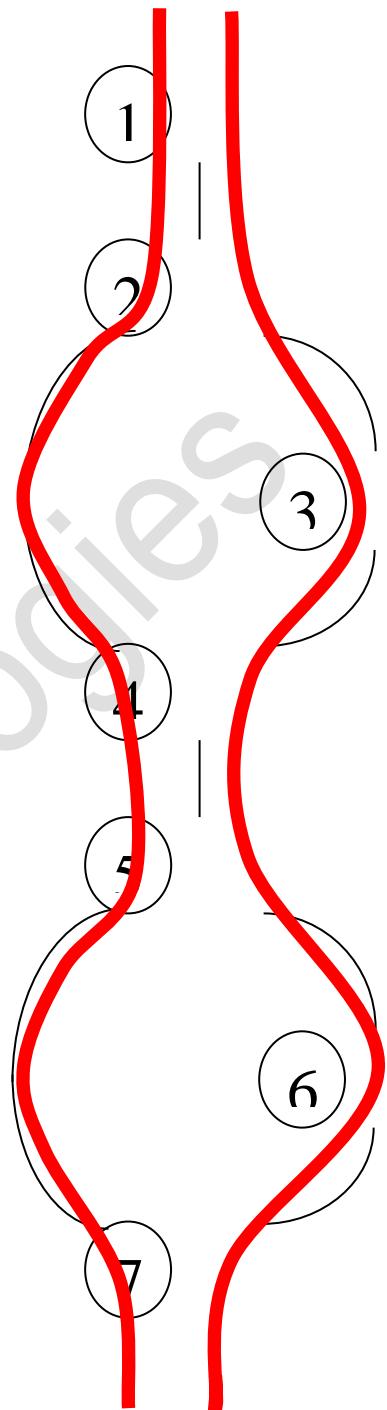
1. Read A
2. If  $A > 40$  Then
3.      $A = A * 2$
4. End If
5. If  $A > 100$  Then
6.      $A = A - 10$
7. End If

- **Answer :**

- Decision/Branch Coverage is 2.

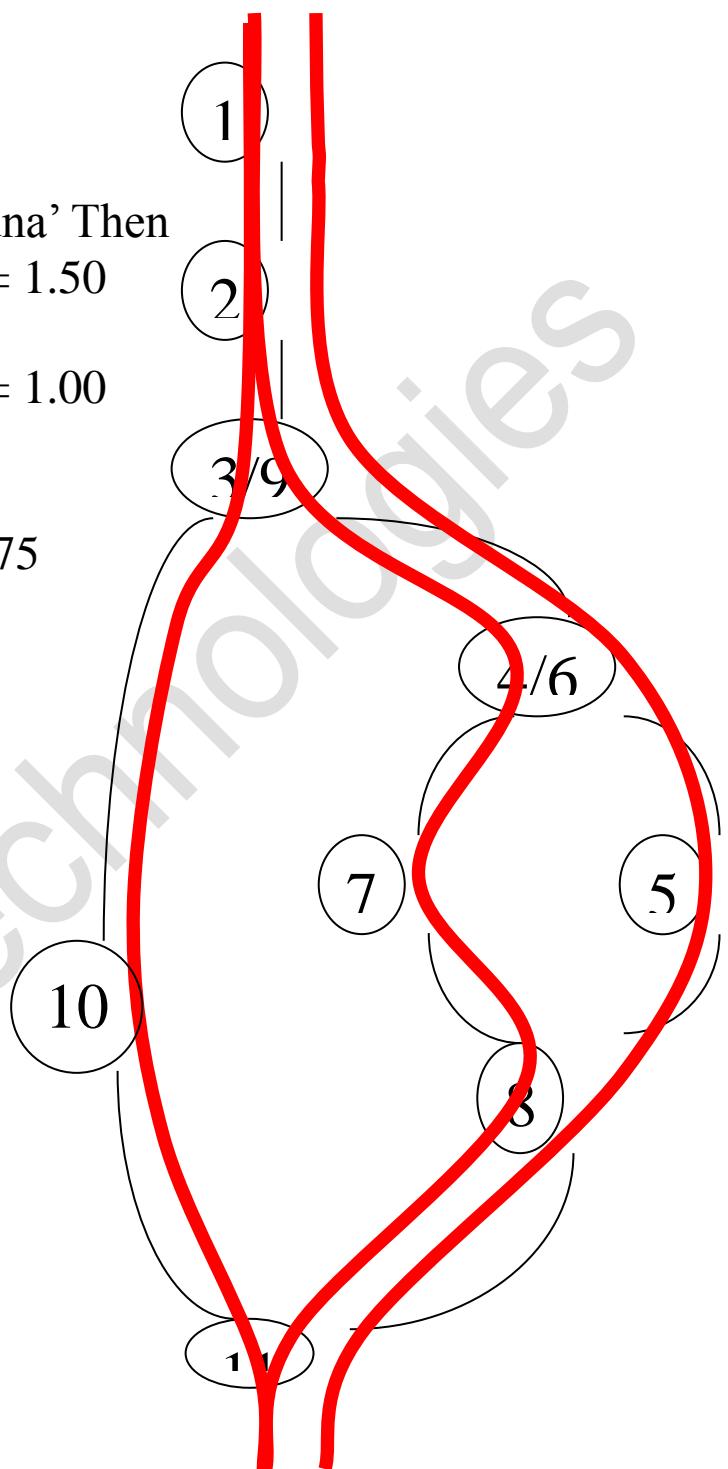
- **Reason:**

- 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



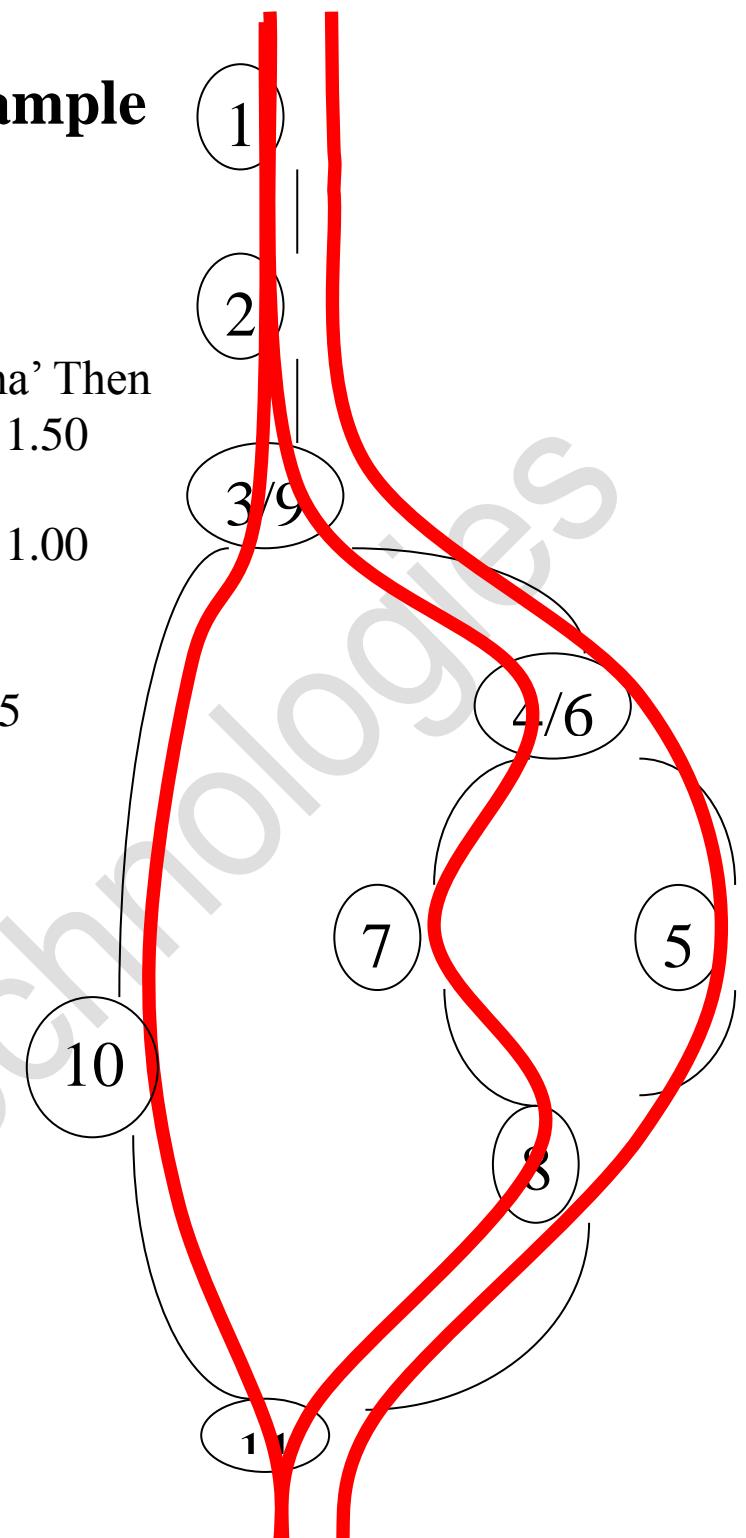
# Statement Coverage Example

```
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**.
- 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.
- **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.

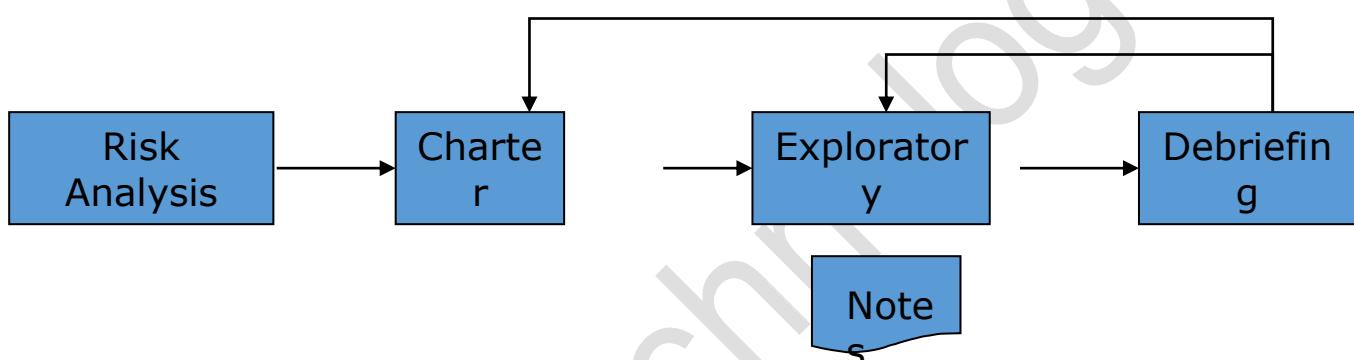
## Types of Adhoc Testing

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

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

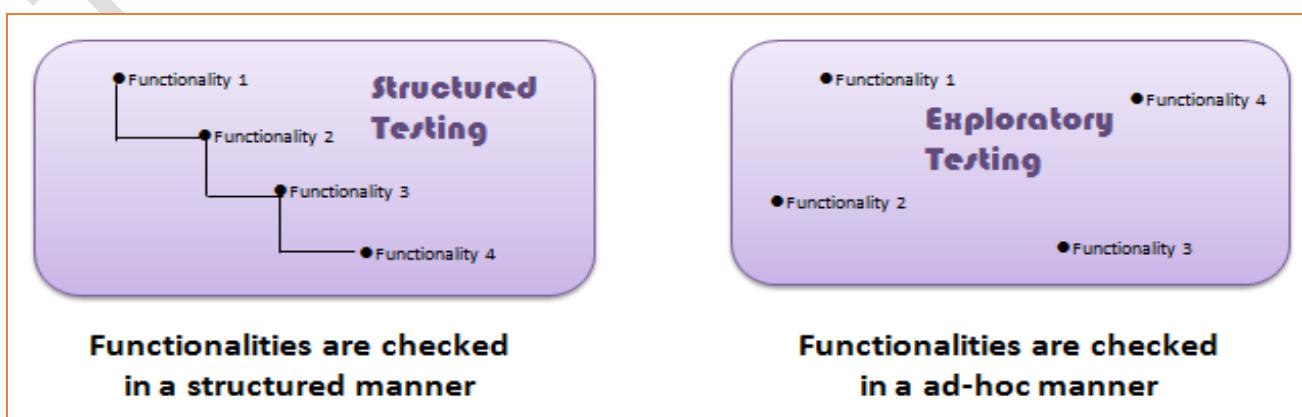
# Exploratory Testing

- 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



**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



# Exploratory vs Scripted Testing

Scripted Testing	Exploratory Testing
Directed from requirements	Directed from requirements and exploring during testing
Determination of test cases well in advance	Determination of test cases during testing
Confirmation of testing with the requirements	Investigation of system or application
Emphasizes on prediction and decision making	Emphasizes on adaptability and learning
Involves confirmed testing	Involves Investigation
Is about Controlling tests	Is about Improvement of test design
Like making a speech – you read from a draft	Like making a conversion – its spontaneous
The script is in control	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

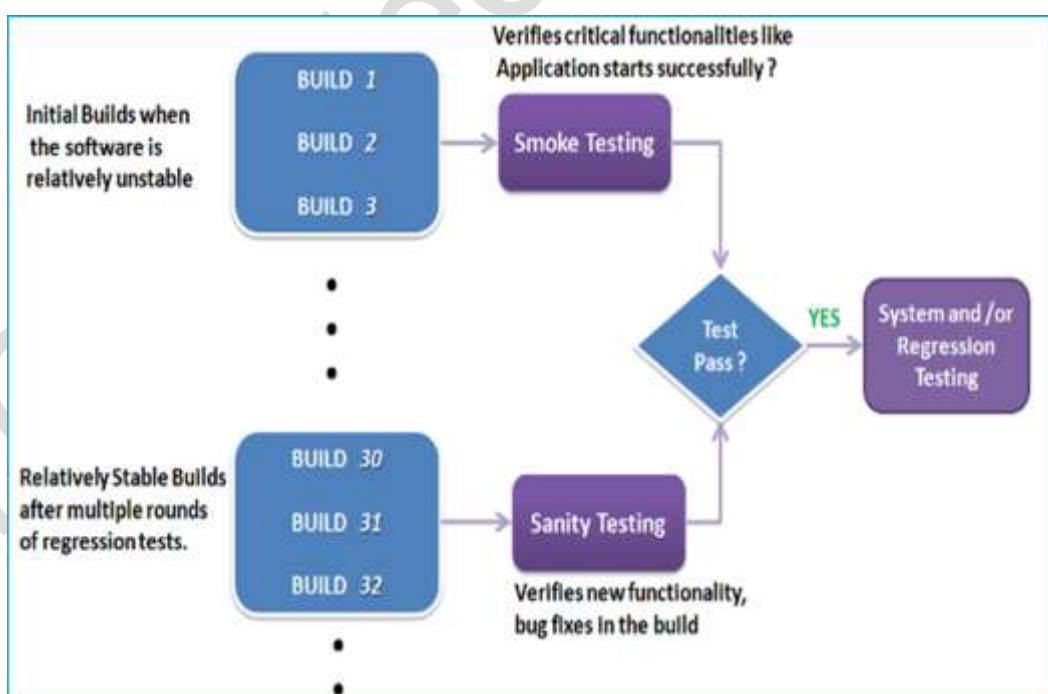
**Techniques**

- 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*"
  - 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.

# 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 you're scientific calculator gives the result of  $2 + 2 = 5!$  Then, there is no point testing the advanced functionalities like  $\sin 30 + \cos 50$ .

# 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	Sanity Testing is done to check the new functionality / bugs have been fixed
The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing	The objective of the testing is to verify the "rationality" of the system in order to proceed with more rigorous testing
This testing is performed by the developers or testers	Sanity testing is usually performed by testers
Smoke testing is usually documented or scripted	Sanity testing is usually not documented and is unscripted
Smoke testing is a subset of Regression testing	Sanity testing is a subset of Acceptance testing
Smoke testing exercises the entire system from end to end	Sanity testing exercises only the particular component of the entire system
Smoke testing is like General Health Check Up	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

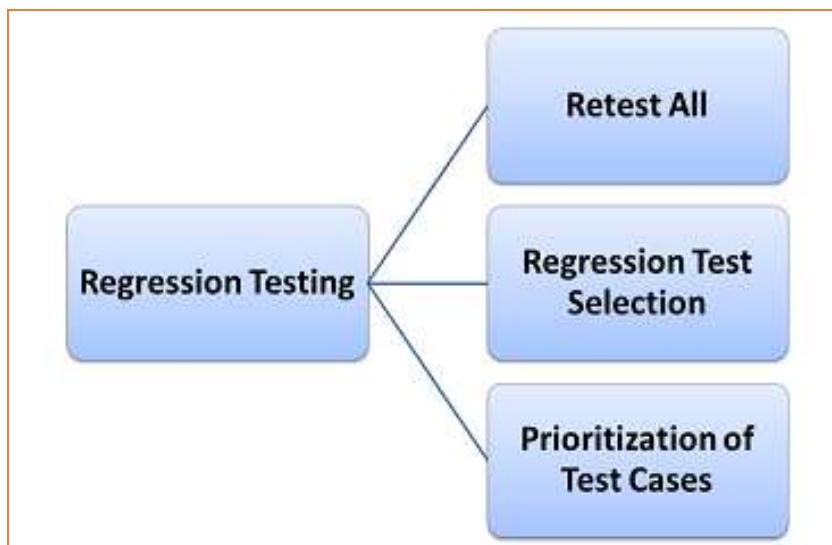
- **Regression testing should be carried out:**
  - when the system is stable and the system or the environment changes
  - when testing bug-fix releases as part of the maintenance phase
- It should be applied at all Test Levels
- It should be considered complete when agreed completion criteria for regression 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:



TOPS Technologies

# 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

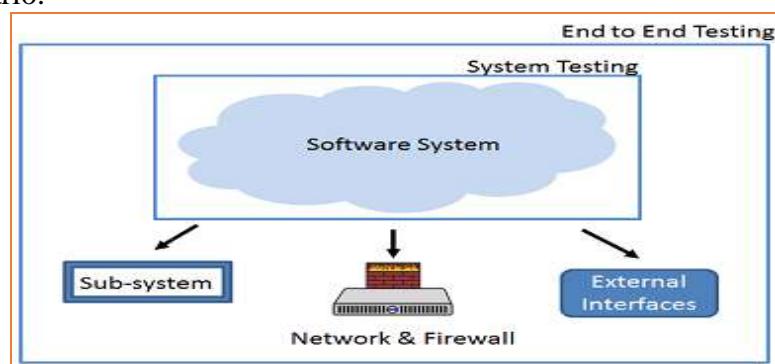
## When use the testing tools?

- 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 re-usable 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.

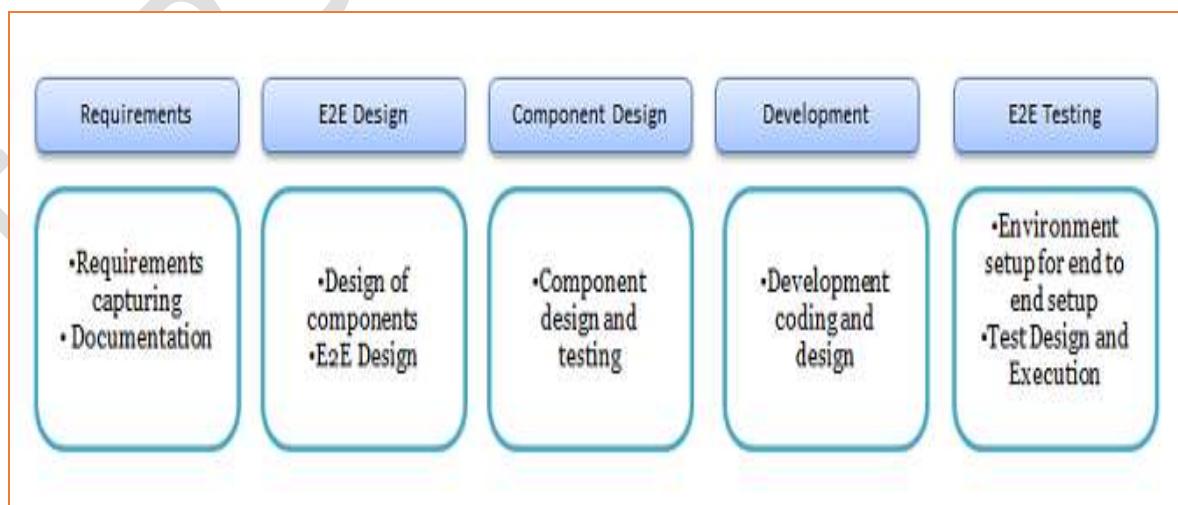
End-to-End testing is also called **Chain Testing**

## 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.

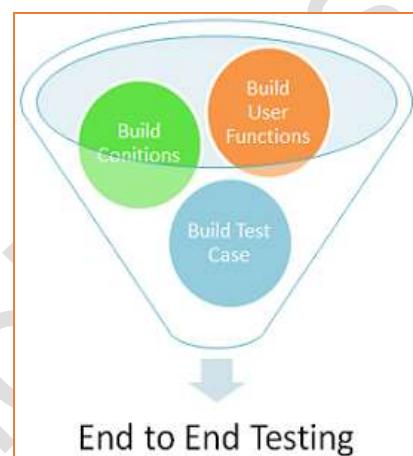
## End-to End Testing Process

- 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 vs System Testing

End to End Testing	System Testing
Validates the software system as well as interconnected sub-systems	Validates just the software system as per the requirements specifications.
It checks the complete end-to-end process flow.	It checks system functionalities and features.
All interfaces, backend systems will be considered for testing	Functional and Non-Functional 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 automate. Hence Manual Testing is preferred.	Both Manual and Automation can be performed for system testing

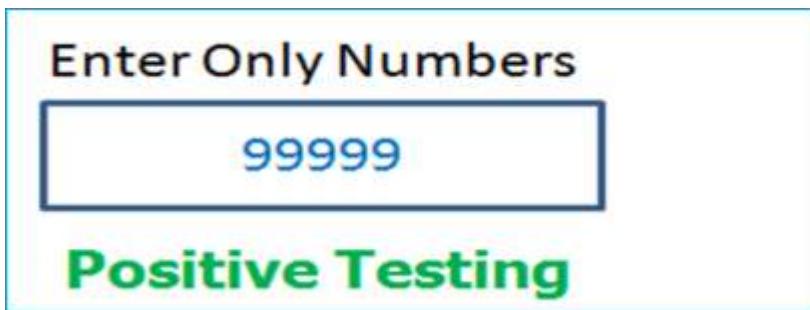
# Positive and Negative Testing

## 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.



## 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

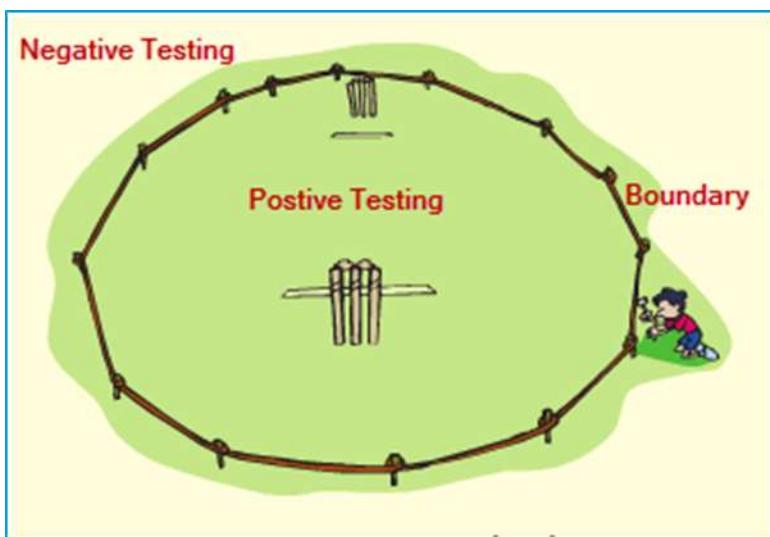
abcdef

**Negative Testing**

TOPS Technologies

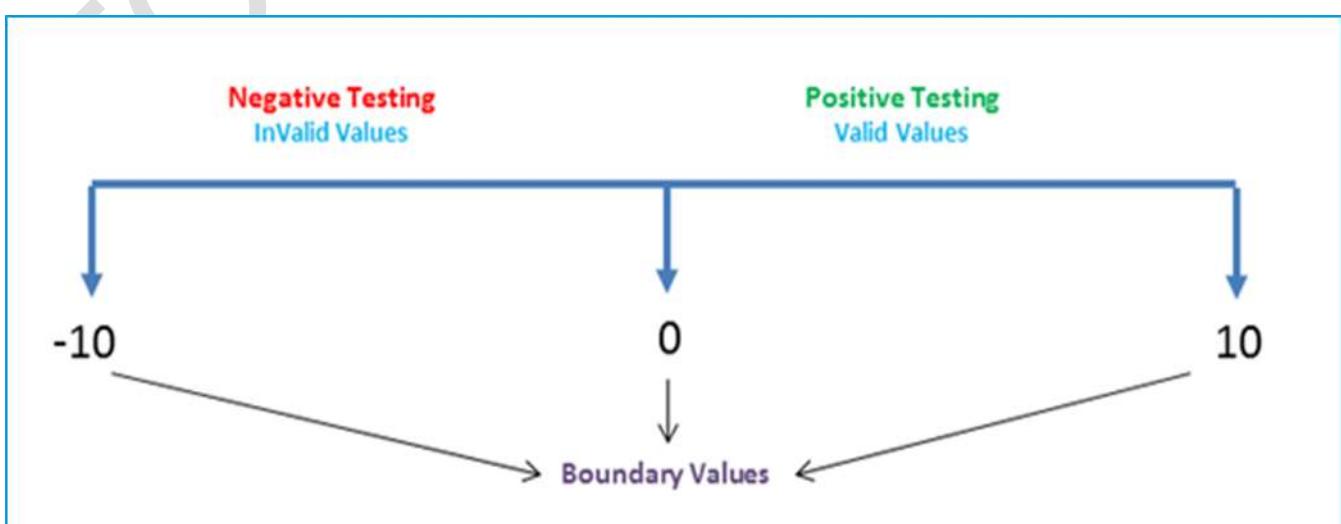
# Boundary Value Analysis (BVA)

- 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.

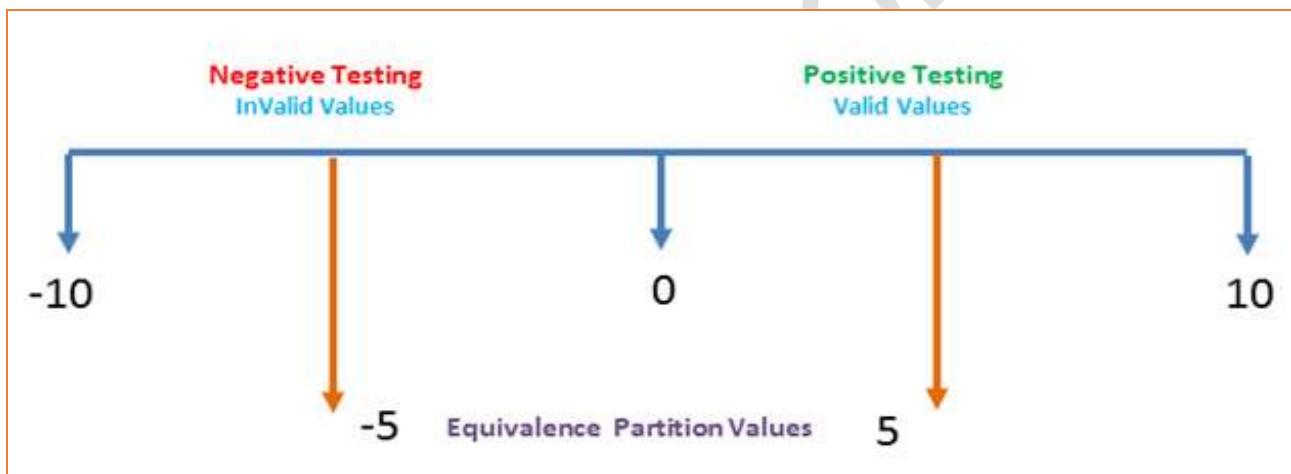


# Equivalence Partitioning (EP)

- 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 scenario.



# 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 and Techniques

## 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 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.
- 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.
- 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 it's 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 – Rework

- 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 access 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.
  - 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:**

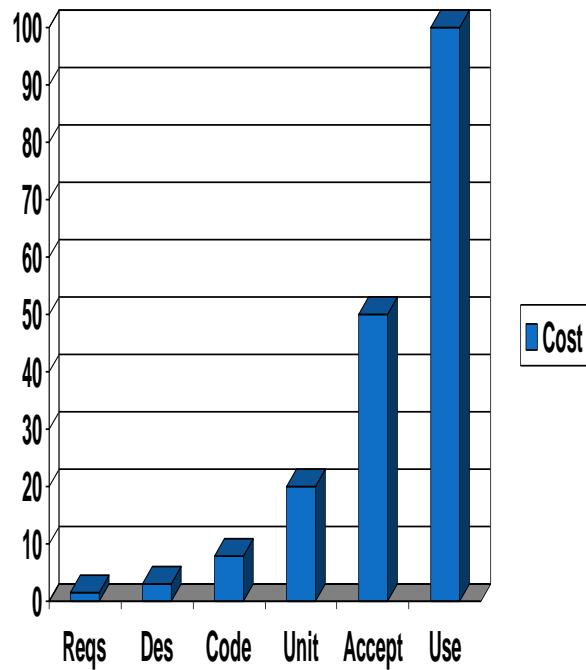
- Delphi Technique
- Analogy Based Estimation
- Test Case Enumeration Based Estimation
- Task (Activity) based Estimation
- IFPUG method

# Why Reviews and Test Process?

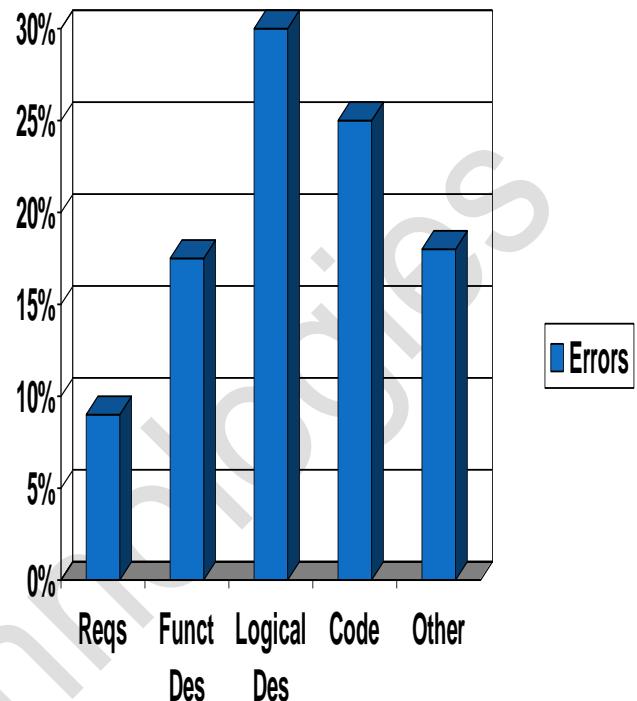
- Improve the specification, design and code
- Education and training of developers and other project staff
- Determine the root causes of defects and measures for preventing recurrence
- Test and improve standards and procedures
- To assess the early stages of development for progress and completeness
- 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

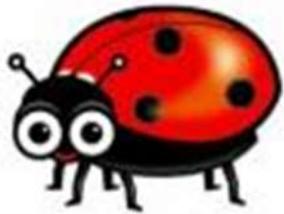


## When and What to Reviews?

- 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

### Introduction

- Agile testing is a software testing practice that follows the rules of the agile manifesto, treating software development as the customer of testing.
- Agile testing involves testing from the customer perspective as early as possible, testing early and often as code becomes available and stable enough from module/unit level testing.
- Since working increments of the software is released very often in agile software development there is also a need to test often.
- This is often done by using automated acceptance testing to minimize the amount of manual labor.
- Agile Testing can begin at the start of the project with continuous integration between development and testing.
- Doing only manual testing in agile development would likely result in either buggy software or slipping schedules because it would most often not be possible to test the whole software manually before every release.
- Agile Testing is not sequential (in the sense its executed only after coding phase) but continuous.
- Agile team works as single team towards a common objective of achieving Quality.
- Agile Testing has shorter time frames called iterations (say from One to Four weeks ).
- This methodology is also called release or delivery driven approach since it gives better prediction on the workable products in short duration of time.

# The Agile Manifesto

- The Agile Manifesto reads, in its entirety, as follows:
- We are uncovering better ways of developing software by doing it and helping others does it. Through this work we have come to value:
  - Individuals and interactions over Processes and tools
  - Working software over Comprehensive documentation
  - Customer collaboration over Contract negotiation
  - Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

## Methodologies of Agile Testing

- There are various methodologies present in agile testing and those are listed below:
  - **Scrum**
  - **eXtreme Programming**
- Below listed methodologies are used less frequently
  - **Dynamic System Development Method (DSDM)**
    - This is an Iterative and incremental approach that emphasizes on the continuous user involvement.
  - **Test Driven Development (TDD)**
    - This is a technique which has short iterations where new test cases covering the desired improvement or new functionality are written first.
  - **Feature Driven Development**
    - This is an iterative and incremental software development process and this can aim depends on the features.
  - **XBreed**
    - Agile enterprise previously known as Xbreed .It is agile way of managing, architecting and monitoring the enterprise.
  - **Crystal**
    - Crystal is an adaptive technique mainly used for software development methodologies.

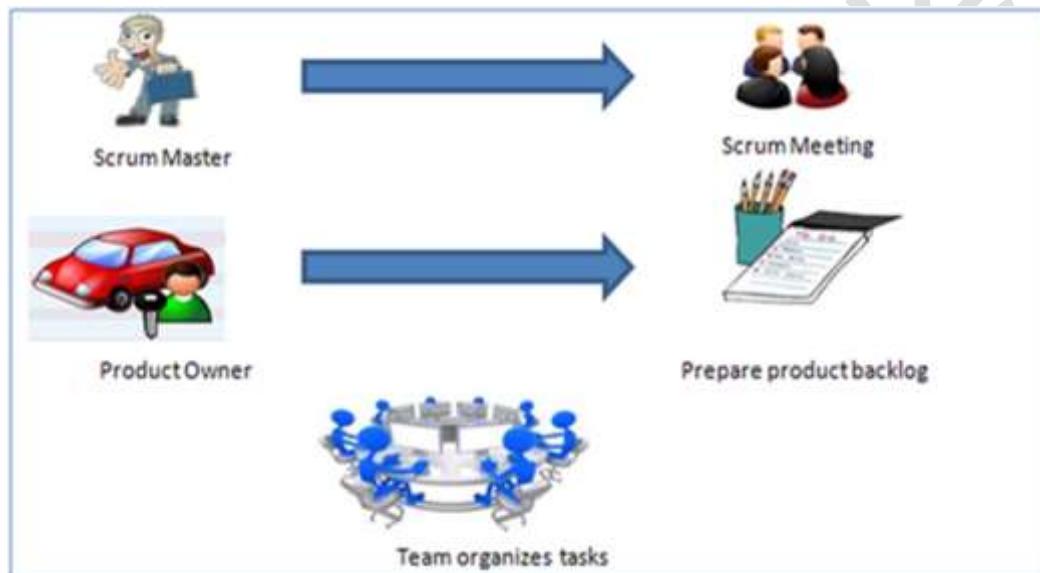
## Scrum

- SCRUM is an agile development method which concentrates particularly on how to manage tasks within a team based development environment. Basically, Scrum is derived from activity that occurs during rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). It consists of three roles and their responsibilities are explained as follows:
  - Scrum Master: Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
  - Product owner: The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
  - Scrum Team: Team manages its own work and organizes the work to complete the sprint or cycle

# Scrum

## Product Backlog

- This is repository where requirements are tracked with details on the no of requirements to be completed for each release.
- It should be maintained and prioritized by scrum master and it should be distributed to the scrum team.
- Team can also request for new requirement addition or modification or deletion



## Phases of Scrum

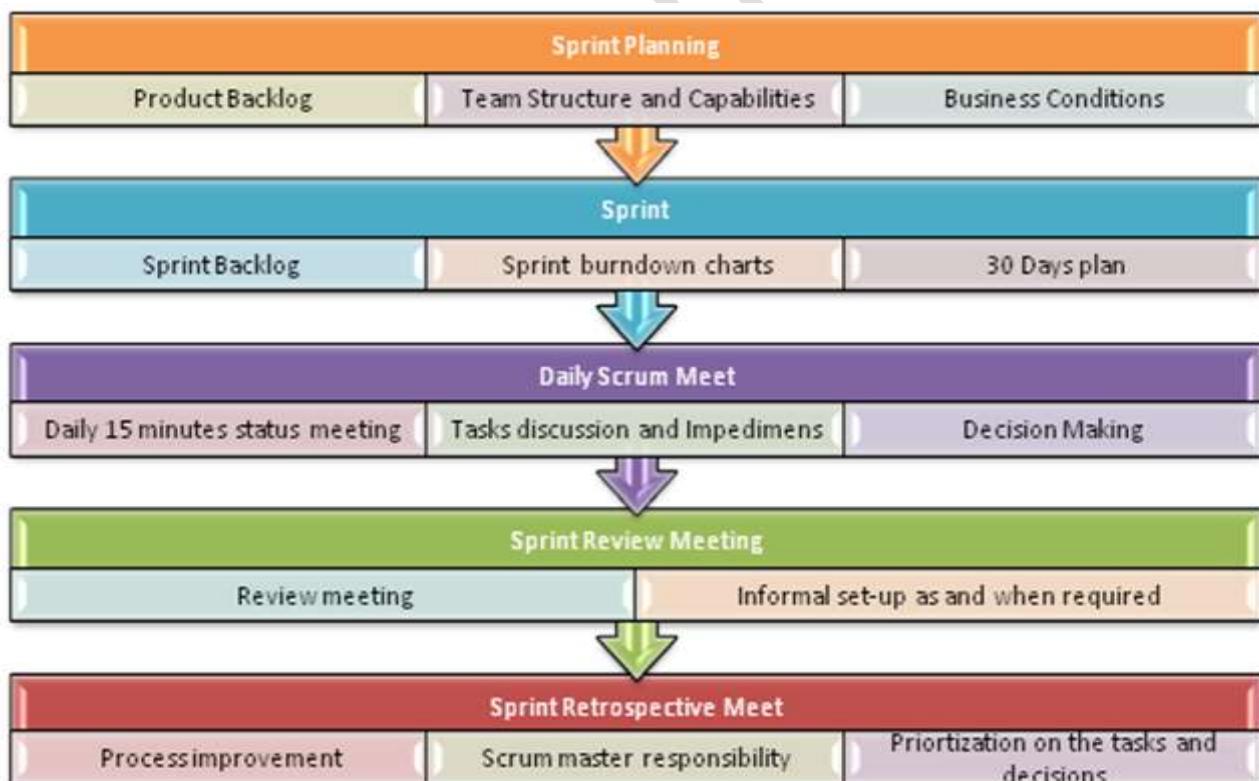
- There are three phases of scrum testing elaborated as follows:
- Test Design
  - User stories are used for development of test cases .User stories are written by the product owner. User stories are short description of functionalities of the System Under Test. Example for Insurance Provider is –Premium can be paid using online system
  - Test Cases are developed based on user stories which should be approved by business analysts and the customers
- Test Execution
  - Test execution is carried out in a lab where both tester and developer work hand in hand.
  - Defect is logged in Defect Management tool which are tracked on a daily basis. Defects can be discussed during scrum meeting
  - Defects are retested as soon as it is fixed and deployed for testing

# Phases of Scrum

- Test Automation
  - This also gains importance due to short delivery timelines
  - Test Automation can be achieved by utilizing various open source or paid tools available in the market.
  - This proves effective in ensuring that everything that needs to be tested was covered. Sufficient Test coverage can be achieved with the close interaction with the team.

# Phases of Scrum

- **Sprint Burn Down Charts**
  - Each day, Scrum Master records the estimated remaining work for the sprint. This is nothing but the Burn Down Chart. It is updated daily.
- **Scrum Practices**
  - Practices are described in detailed:



# Process Flow of Scrum

- Process flow of scrum testing is as follows:
- Each iteration of a scrum is known as Sprint
- Product backlog is a list where all details are entered to get end product
- During each Sprint, top items of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of sprint, team delivers product functionality

# eXtreme Programming

- This is a light weight agile testing methodology in which development and testing happen in parallel. Business requirements are gathered in terms of stories.
- All those stories are stored in a place called parking lot.
- In this type of methodology, releases are based on the shorter cycles called Iterations with span of 14 days' time period.
- Each iteration include phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.

# eXtreme Programming Phases

- **Planning**
  - Identification of stakeholders and sponsors
  - Infrastructure Requirements
  - Security related information and gathering
  - Service Level Agreements and its conditions
- **Analysis**
  - Capturing of Stories in Parking lot
  - Prioritize stories in Parking lot
  - Scrubbing of stories for estimation
  - Define Iteration SPAN(Time)
  - Resource planning for both Development and QA teams
- **Design**
  - Break down of tasks
  - Test Scenario preparation for each task
  - Regression Automation Framework

# eXtreme Programming Phases

- **Execution**

- Coding
- Unit Testing
- Execution of Manual test scenarios
- Defect Report generation
- Conversion of Manual to Automation regression test cases
- Mid Iteration review
- End of Iteration review

- **Wrapping**

- Small Releases
- Regression Testing
- Demos and reviews
- Develop new stories based on the need
- Process Improvements based on end of iteration review comments

- **Closure**

- Pilot Launch
- Training
- Production Launch
- SLA Guarantee assurance
- Review SOA strategy
- Production Support
- There are two story boards available to track the work on a daily basis and those are listed below for reference.
- Story Cardboard
  - This is traditional way of collecting all the stories in a board in the form of stick notes to track daily XP activities. As this manual activity involves more effort and time, it is better to switch to online form.
- Online Story board
  - Online tool Storyboard can be used to store the stories .This can be used by several teams for different purposes.

# Scrum vs eXtreme Programming

XP Testing	Scrum Testing
Engineering practices like test driven development, refactoring, pair programming, etc...	Managing of requirements or requested features
Stories are used as single line requirements	User Stories are used as requirements
Stories are stored in Parking lot	Stories are stored in Product backlog and Sprint backlog
Iteration is used in XP testing	Iteration is termed as Sprint
Iteration period is for 14 days max	Iteration period is for 30 days
Requirement changes are accepted during iteration	Requirements changes are acceptable after the current iteration
Mid and end of Iteration review meetings	Daily scrum meeting

## Agile Metrics

- Drag Factor
  - Effort in hours which do not contribute to sprint goal
  - Drag factor can be improved by reducing number of shared resources, reducing the amount of non-contributing work
  - New estimates can be increased by percentage of drag factor -New estimate = (Old estimate + drag factor)
- Velocity
  - Amount of backlog converted to shippable functionality of sprint
- No of Unit Tests added
- Time taken to complete daily build
- Bugs detected in an iteration or in previous iterations
- Production defect leakage

## 12 Principles of Agile Methods

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily cooperation between business people and developers
7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstances

## Advantages of Agile Testing

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

## Disadvantages of Agile Testing

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

# Model - 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.
- 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.

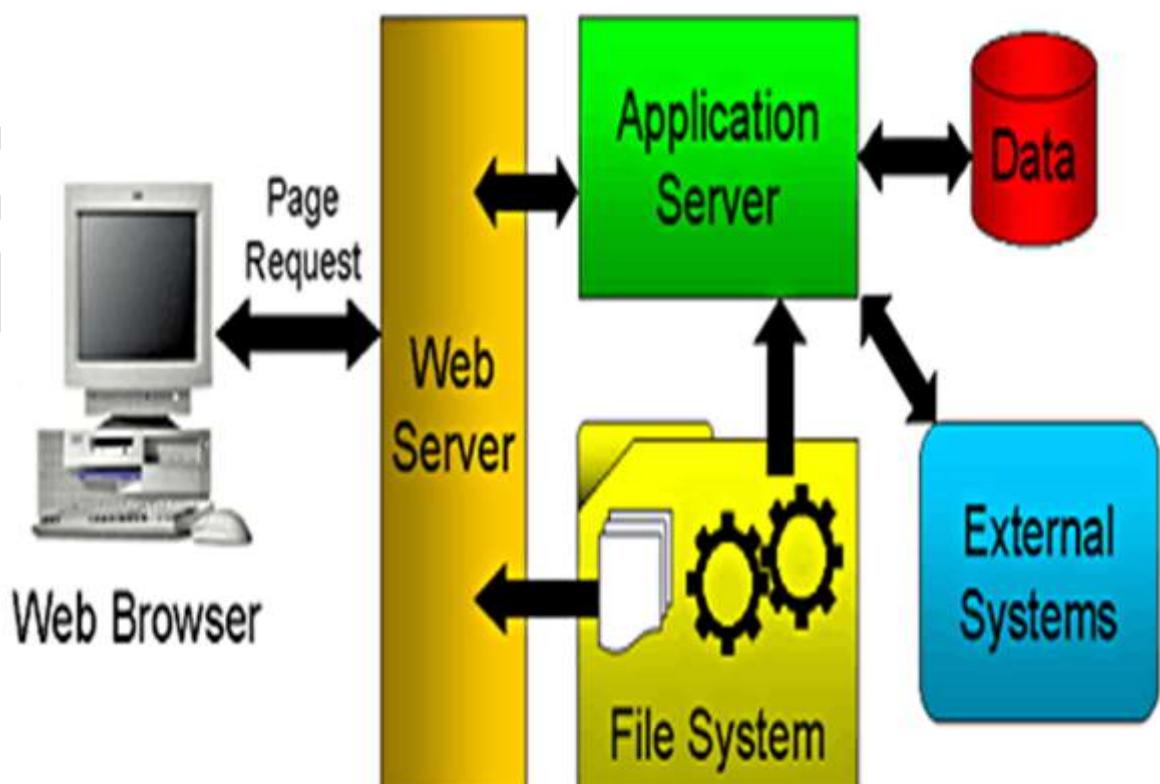
# Introduction

- 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 type 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++, D2K, PowerBuilder etc.,
- The backend for these applications would be MS Access, SQL Server, Oracle, Sybase, Mysql, Quad base

# 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 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

# Web Testing Guide Lines

- **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.
- **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
- **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
  - Colors of text, foreground and background
  - Scalability to fit paper size
  - Tables and borders
- **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.

- **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.

# Web Testing Guide Lines

- **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)

- **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

## Introduction

- In usability testing, you'll be looking at aspects of your web application that affect the user's experience, such as:
  - How easy is it to navigate through your web application?
  - Is it obvious to the user which actions are available to him or her?
  - Is the look-and-feel of your web application consistent from page to page, including font sizes and colors?
  - If a user forgets to fill in a required field, you might think it is a good idea to present the user with a friendly error message and change the color of the field label to red or some other conspicuous color.
  - However, changing the color of the field label would not really help a user who has difficulty deciphering colors. The use of color may help most users, but you would want to use an additional visual clue, such as placing an asterisk beside the field in question or additionally making the text bold.
- In Usability Testing, a small-set of target end-users, of a software system, “use” it to expose usability defects.
- This testing mainly focuses on the user's-ease to use the application, flexibility in handling controls and ability of the system to meet its objectives.
- This testing is recommended during the initial design phase of SDLC, which gives more visibility on the expectations of the users.
- There are two methods available to do usability testing –
  - Laboratory Usability Testing
  - Remote 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.

# 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.
  - N Number of systems can be connected and compare the results.

## GUI Testing

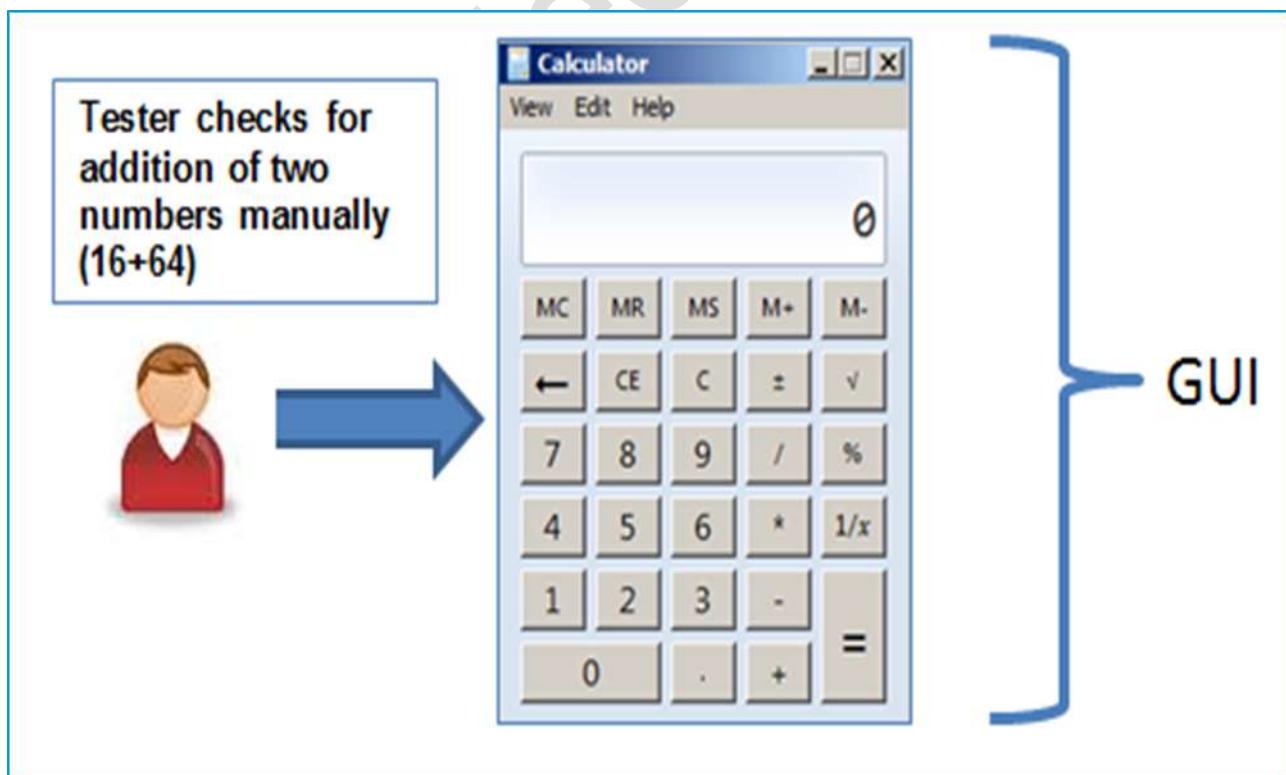
### Introduction

- 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.

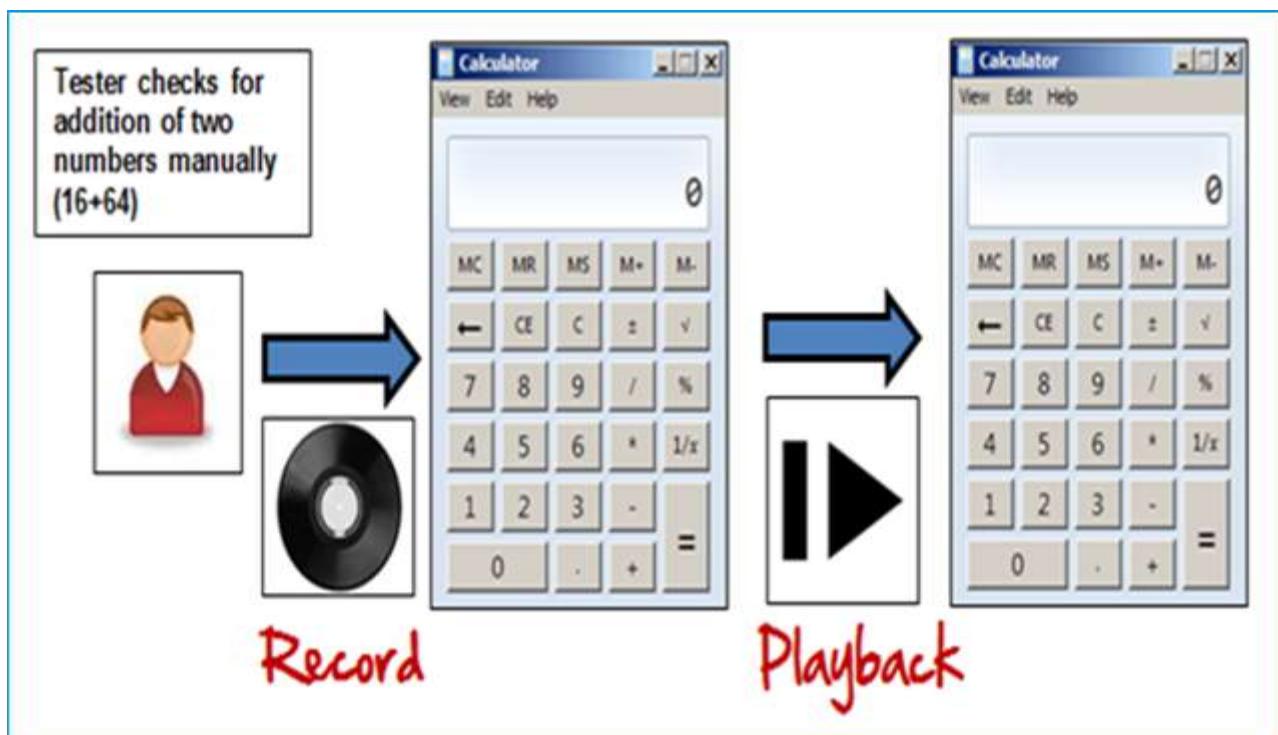
# Approach of GUI Testing

- **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

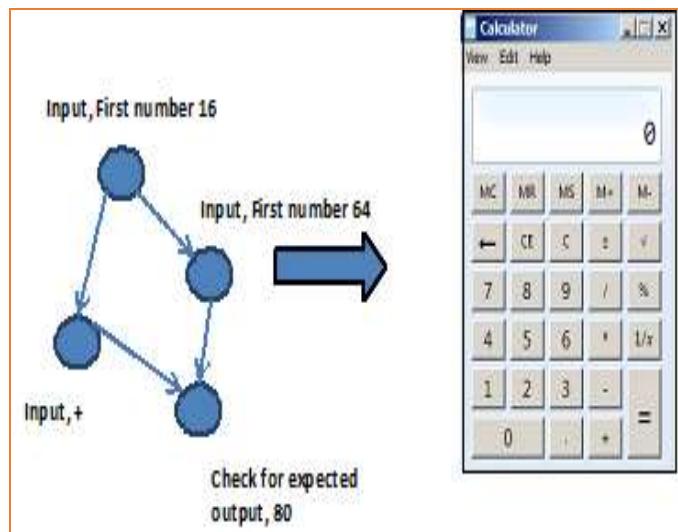


# Record & Play in GUI



# 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 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.

# 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
<b>Requirements</b>	Security analysis for requirements and check abuse/misuse cases
<b>Design</b>	Security risk analysis for designing. Development of test plan including security tests
<b>Coding and Unit Testing</b>	Static and Dynamic Testing and Security white box testing
<b>Integration Testing</b>	Black Box Testing
<b>System Testing</b>	Black Box Testing and Vulnerability scanning
<b>Implementation</b>	Penetration Testing, Vulnerability Scanning
<b>Support</b>	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.

# 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.
  
- **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.

# 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.**
- 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 Tools

- Neo Load
- App Perfect

TOPS Technologies

# 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"> <li>• Server Machines</li> <li>• Processors</li> <li>• Memory</li> <li>• Disk Storage</li> <li>• Load Machines configuration</li> <li>• Network configuration</li> </ul>	<ul style="list-style-type: none"> <li>• Operating System</li> <li>• Server Software</li> </ul>

# 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 Tools

- Load runner
- 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.	Stress testing is to test the system behavior under extreme conditions and is carried out till the system failure.
Load testing identifies the bottlenecks in the system under various workloads and checks how the system reacts when the load is gradually increased	Stress testing determines the breaking point of the system to reveal the maximum point after which it breaks.
Load testing does not break the system	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
- 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

# 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
- **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.

TOPS Technologies

# 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

- **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.

- **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.
- **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 user's 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.
- **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
- **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.

TOPS Technologies

# 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?
- **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.
- **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?
- **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
- **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.
- **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

- **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

- **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 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
- **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
- **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.

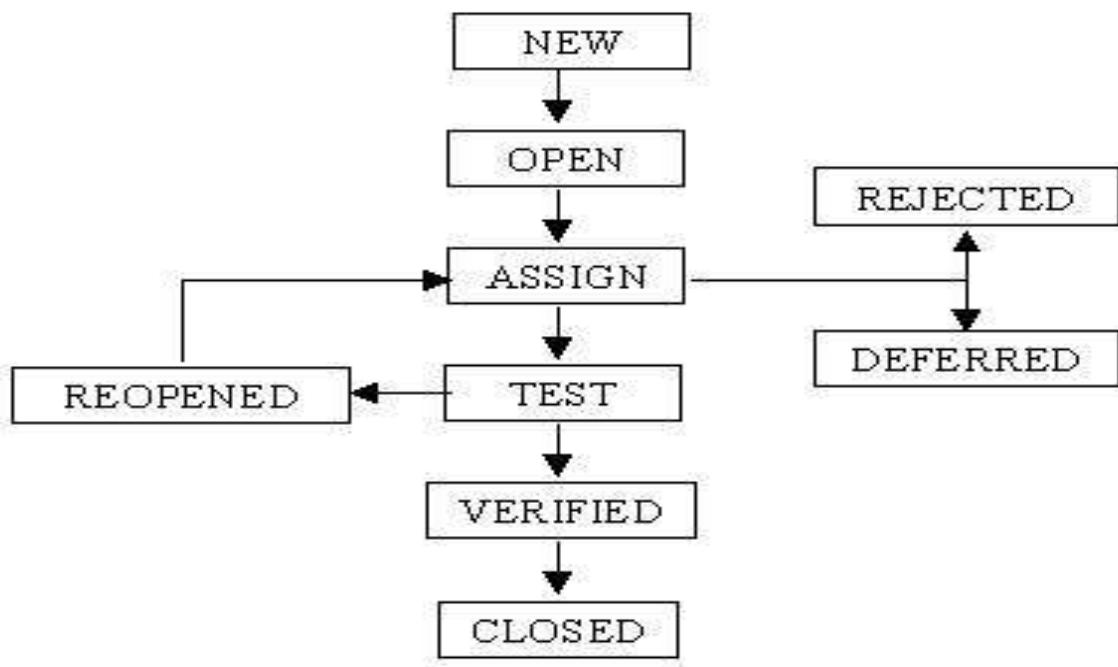
- Examples:
  - Authentication: Accepting an invalid username/password
  - Authorization: Accessibility to pages though permission not given

- **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 – P0, P1, P2, P3, P4, P5 (P0-Highest, P5-Lowest)

- 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.

TOPS Technologies

# 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.
- **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 field 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 are-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 Management and 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:



TOPS\_tech\_vlog

## Module - 5 Bugzilla

### 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

Welcome, tops <tops@tops-int.com>

You are seeing this page because some of the core parameters have not been set up yet. The goal of this page is to inform you about the last steps required to set up your installation correctly.

As an administrator, you have access to all administrative pages, accessible from the [Administration](#) link visible at the bottom of this page. This link will always be visible, on all pages. From there, you must visit at least the [Parameters](#) page, from where you can set all important parameters for this installation, among others:

- [\\$urlbase](#), which is the URL pointing to this installation and which will be used in emails (which is also the reason you see this page: as long as this parameter is not set, you will see this page again and again).
- [\\$cookiepath](#) is important for your browser to manage your cookies correctly.
- [\\$maintainer](#), the person responsible for this installation if something is running wrongly.

Also important are the following parameters:

- [requirelogin](#), if turned on, will protect your installation from users having no account on this installation. In other words, users who are not explicitly authenticated with a valid account cannot see any data. This is what you want if you want to keep your data private.
- [createemailaccount](#) defines which users are allowed to create an account on this installation. If set to ".\*" (the default), everybody is free to create his own account. If set to "@mycompany.com", only users having an account @mycompany.com will be allowed to create an account. If left blank, users will not be able to create accounts themselves; only an administrator will be able to create one for them. If you want a private installation, you must absolutely set this parameter to something different from the default.
- [mail\\_delivery\\_method](#) defines the method used to send emails, such as sendmail or SMTP. You have to set it correctly to send emails.

After having set up all this, we recommend looking at Bugzilla's other parameters as well at some time so that you understand what they do and whether you want to modify their settings for your installation.

[Home](#) | [New](#) | [Browse](#) | [Search](#) | [Search](#) | [Reports](#) | [Preferences](#) | [Administration](#) | Log out tops@tops-int.com

My Bugs

# Bugzilla : Add User

The screenshot shows a Mozilla Firefox browser window with the following details:

- Toolbar:** File, Edit, View, History, Bookmarks, Tools, Help.
- Address Bar:** Select user, Survey Index Page, Edit Groups, Edit Workflow. The URL is http://localhost:5050/editusers.cgi?action=list&matchvalue=login\_name&matchstr=&matchtype=substr&matchstr=
- Search Bar:** Search
- Content Area:**
  - Bugzilla – Select user**
  - Home | New | Browse | Search | Search | [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com
  - 2 users found.
  - A table with three columns: Edit user..., Real name, and Account History.

Edit user...	Real name	Account History
<a href="#">tops@tops-int.com</a>	tops	<a href="#">View</a>
<a href="#">vishal.shah@tops-int.com</a>	Vishal	<a href="#">View</a>

  - If you do not wish to modify a user account at this time, you can [find other users](#) or [add a new user](#).
  - My Bugs
- Taskbar:** Start button, Mozilla Firefox icon, Address bar showing the same URL, and system icons.

TOPS

# Bugzilla : Search User

The screenshot shows a Mozilla Firefox browser window with four tabs open:

- Selected tab: "Select user" (http://localhost:5050/editusers.cgi?action=list&matchvalue=login\_name&matchstr=&mat=)
- "Survey Index Page"
- "Edit Groups"
- "Edit Workflow"

The main content area displays the "Bugzilla - Select user" page. It shows a table with two users found:

Edit user...	Real name	Account History
<a href="#">tops@tops-int.com</a>	tops	<a href="#">View</a>
<a href="#">vishal.shah@tops-int.com</a>	Vishal	<a href="#">View</a>

A message at the bottom of the page says: "If you do not wish to modify a user account at this time, you can [find other users](#) or [add a new user](#)".

The browser's address bar shows the URL: http://localhost:5050/editusers.cgi?action=edit&userid=2&matchvalue=login\_name&groupid=1&grouprestrict=&matchtype=substr&matchstr=

TOPS

# Select User

File Edit View History Bookmarks Tools Help

Select user Survey Index Page Edit Groups Edit Workflow

http://localhost:5050/editusers.cgi?action=list&matchvalue=login\_name&matchstr=&mat=0&matchtype=substr&matchstr=

Bugzilla - Select user

Home | New | Browse | Search | Search [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com

2 users found.

Edit user...	Real name	Account History
<a href="#">tops@tops-int.com</a>	tops	<a href="#">View</a>
<a href="#">vishal.shah@tops-int.com</a>	Vishal	<a href="#">View</a>

If you do not wish to modify a user account at this time, you can [find other users](#) or [add a new user](#).

Home | New | Browse | Search | Search [?] | Reports | Preferences | Administration | Help | Log out tops@tops-int.com

My Bugs

http://localhost:5050/editusers.cgi?action=edit&userid=2&matchvalue=login\_name&groupid=1&grouprestrict=0&matchtype=substr&matchstr=

start Mozilla F... 11:27 AM

# Edit User

The screenshot shows a Microsoft Internet Explorer window with the title bar "User vishal.shah@tops-int.co...". The address bar shows "http://localhost:5055/editusers.cgi". The main content area displays a success message: "Bugzilla – User vishal.shah@tops-int.com updated" followed by "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.

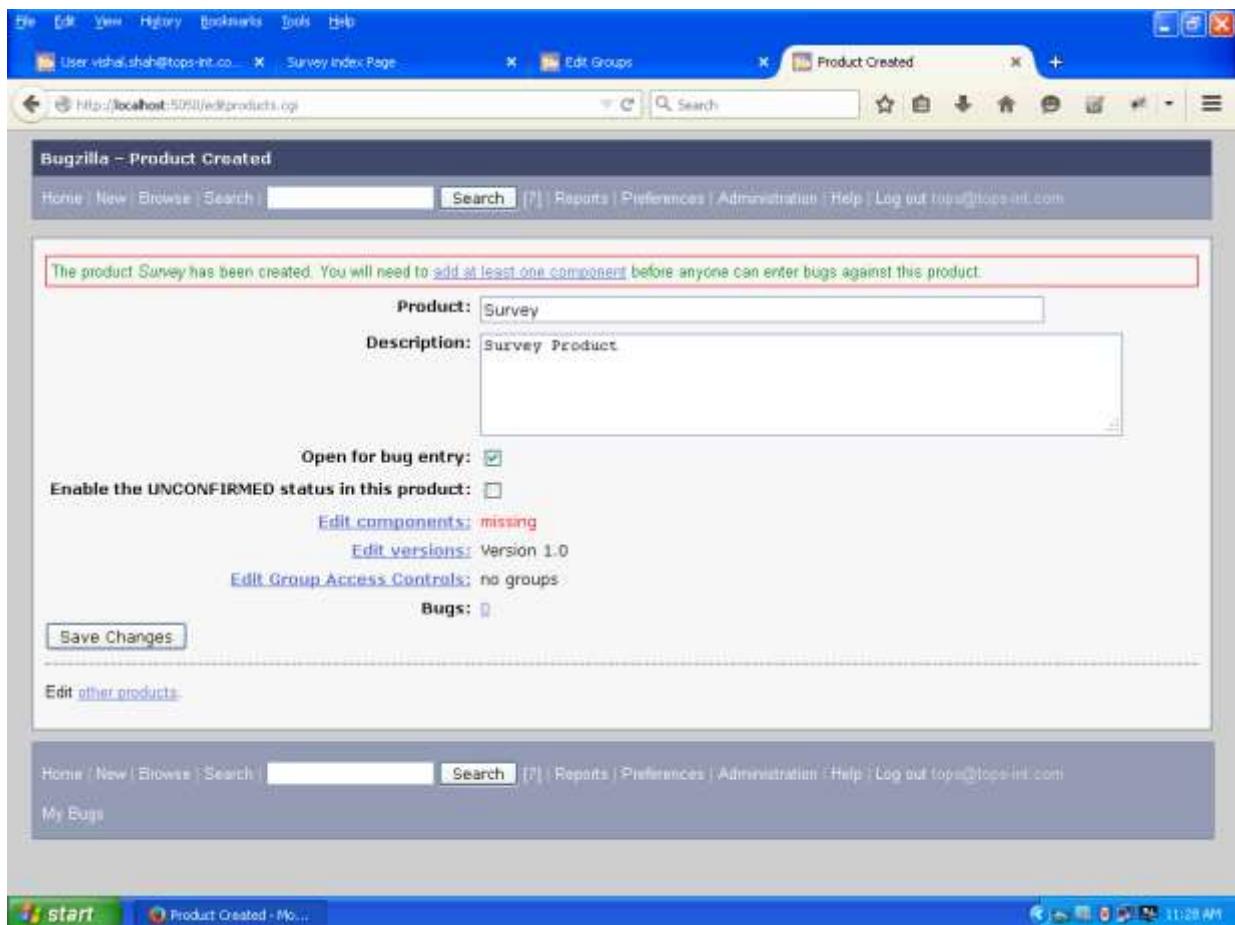
". Below this, there are input fields for "Login name" (vishal.shah@tops-int.com), "Real name" (Vishal), and "Password" (empty). A checkbox for "Bugmail Disabled" is checked. There is also a "Disable text" field which is empty. Under "Group access", there is a list of checkboxes for various permissions, with "admin: Administrators" checked. The status bar at the bottom shows "User vishal.shah@tops-int.com" and the time "11:28 AM".

# Add product

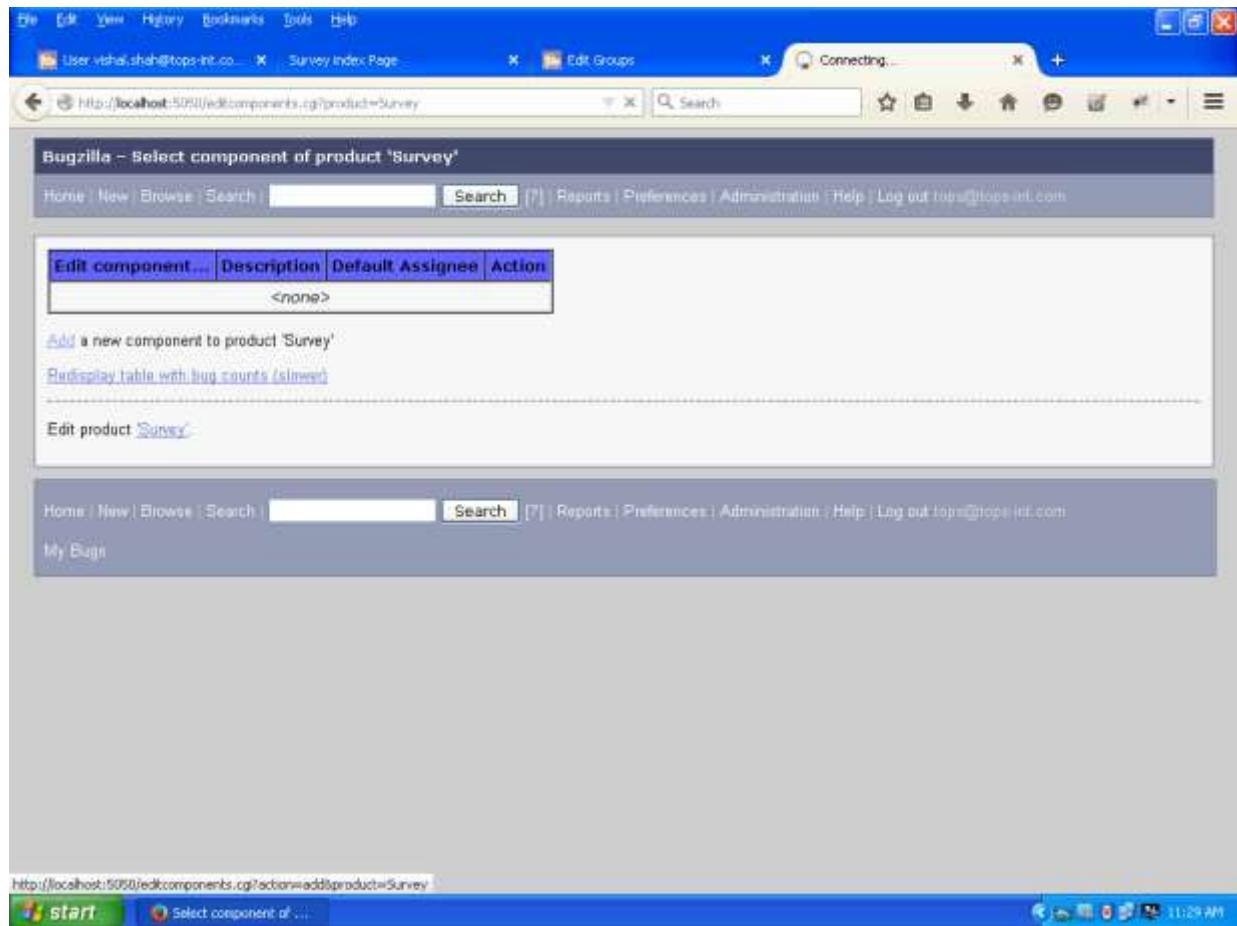
The screenshot shows a Mozilla Firefox browser window with the following details:

- Title Bar:** File Edit View History Bookmarks Tools Help
- Address Bar:** User vishal.shah@tops-int.co... Survey Index Page Add Product http://localhost:50211/edl/products.cgi?action=add
- Toolbar:** Back Forward Stop Refresh Home Stop Search
- Content Area:**
  - Bugzilla – Add Product**
  - Form Fields:**
    - Product: Survey
    - Description: (Empty text area)
    - Open for bug entry:
    - Enable the UNCONFIRMED status in this product:
    - Version: unspecified
    - Create chart datasets for this product:
  - Buttons:** Add
  - Links:** Edit other products
- 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 Product - Mozilla ... 11:28 AM

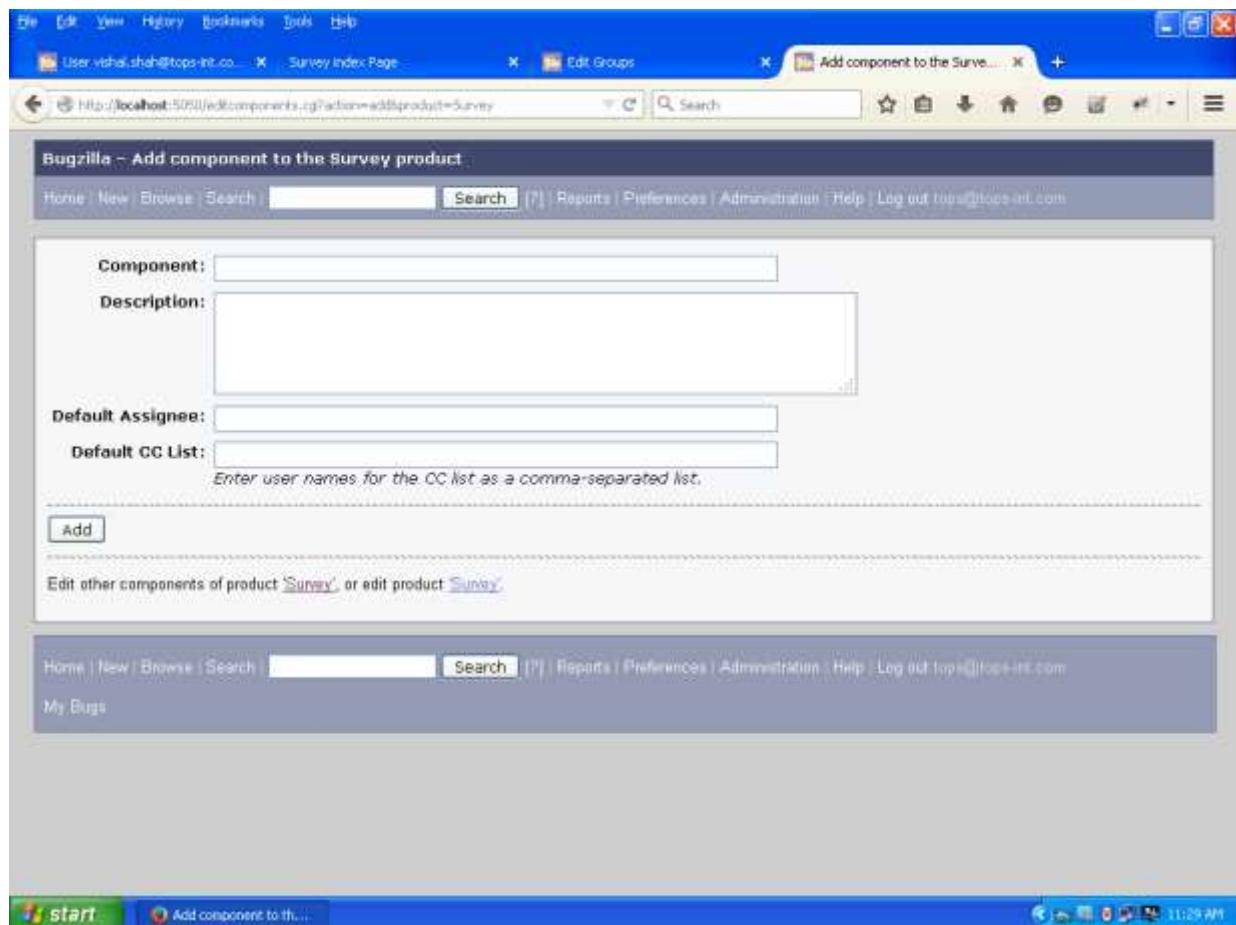
# Product Added



# Select component on product



# Add component to product



# Component Added

The screenshot shows a web browser window with three tabs open:

- User vishal.shah@tops-int.co... - Survey Index Page
- Edit Groups
- Component Created

The active tab is "Component Created". The URL in the address bar is <http://localhost:50201/editComponents.cgi>. The page title is "Bugzilla - Component Created".

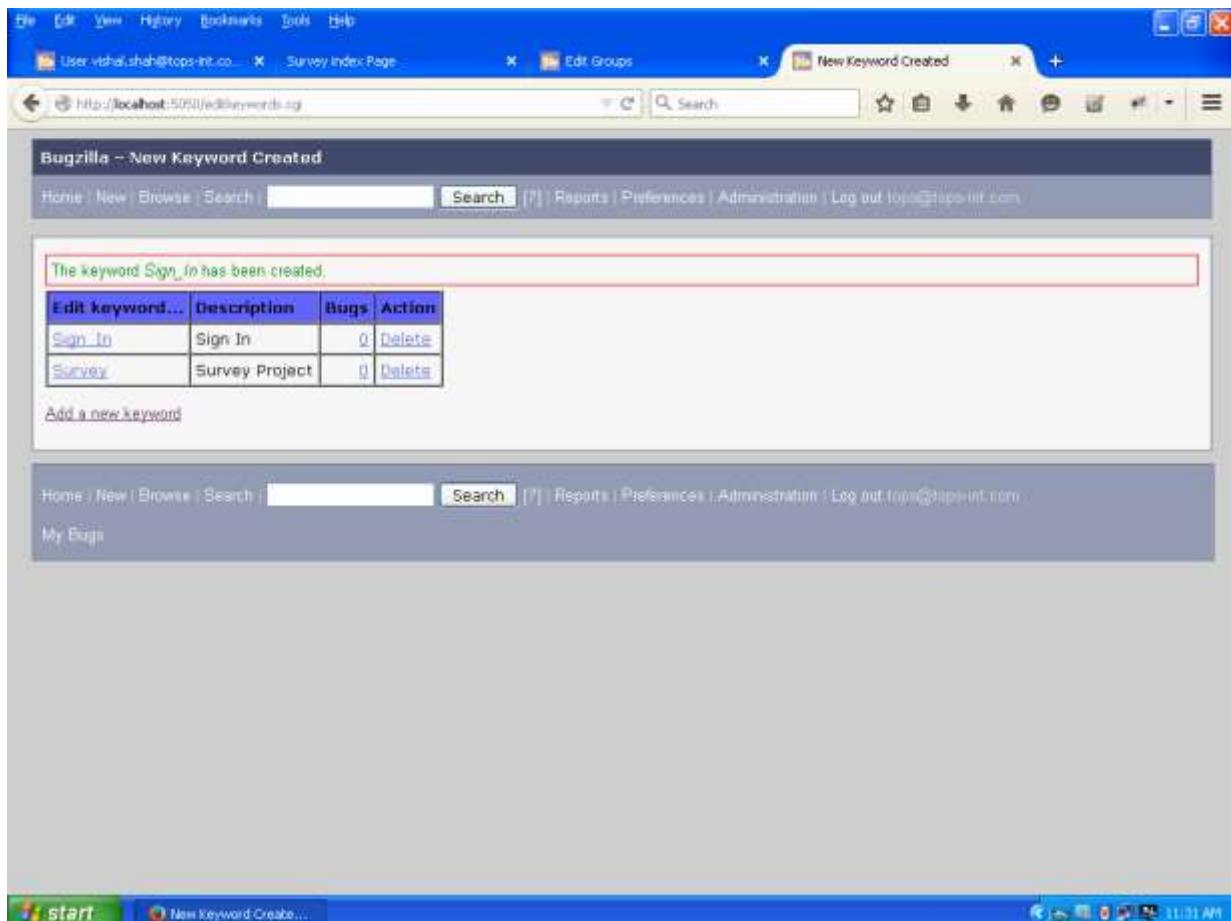
The main content area displays a message: "The component 'Sign In' has been created." Below this, there is a table:

Edit component...	Description	Default Assignee	Action
<a href="#">Sign In</a>	Sign In	tops@tops-int.com	<a href="#">Delete</a>

Below the table, there are links: "Add a new component to product 'Survey'", "Recent day table with bug counts (showed)", and "Edit component [Sign In](#) or edit product [Survey](#)".

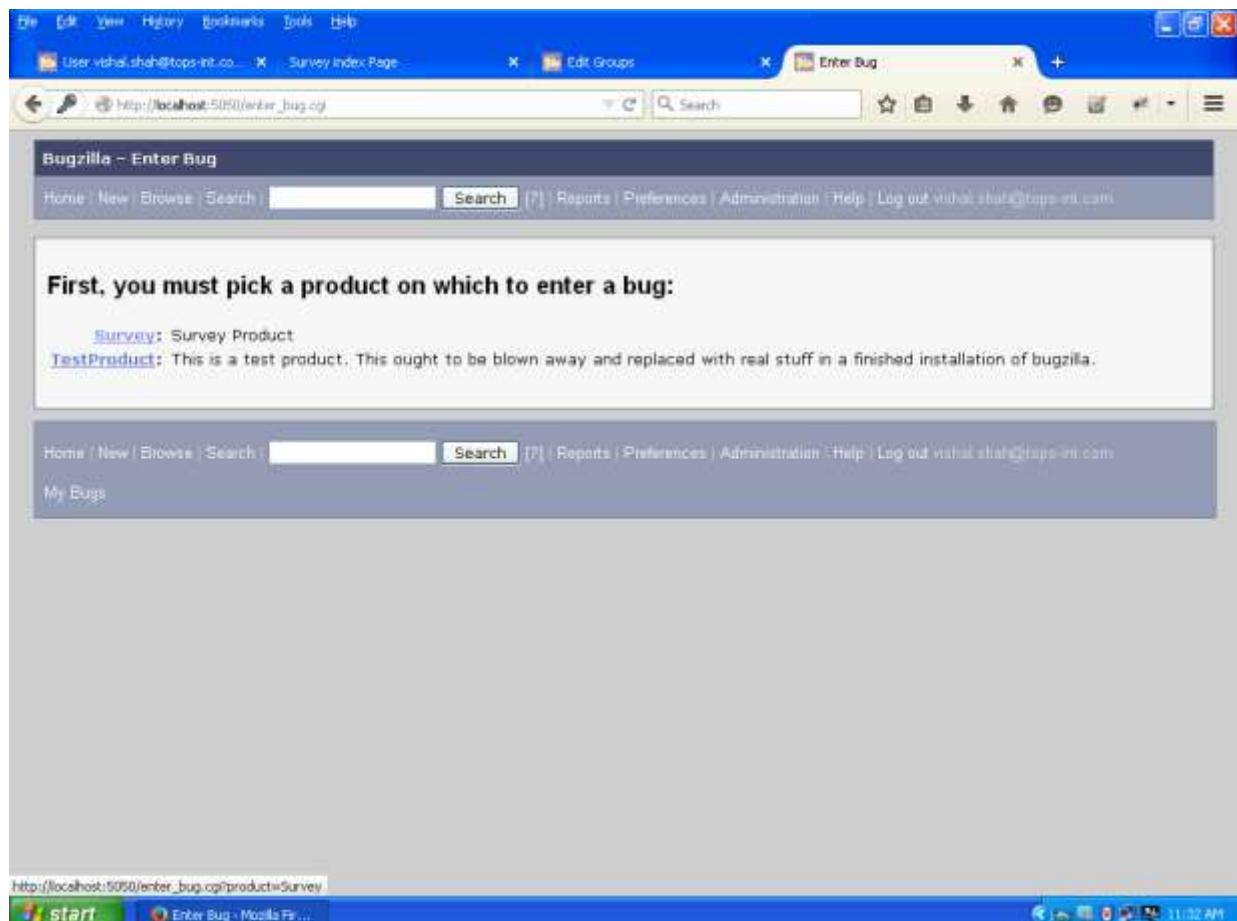
The browser's status bar at the bottom shows the URL <http://localhost:50201/editComponents.cgi?componentID=1&productID=1>, the title "Component Created", and the time "11:29 AM".

# New Keyword Added



TOPS

# Select Product before enter new Bug



# New Bug entry Main Page

The screenshot shows a Microsoft Internet Explorer window with the title "Enter Bug: Survey". The URL in the address bar is "http://localhost:5050/enter\_bug.cgi?product=Survey". The page content is as follows:

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.

**Show Advanced Fields** (\* = Required Field)

**Product:** Survey      **Reporter:** vishal.shah@tops-int.com

**\* Component:** Sign In  
Sign Up  
Sign Up User Registration

**Component Description:**  
Sign In

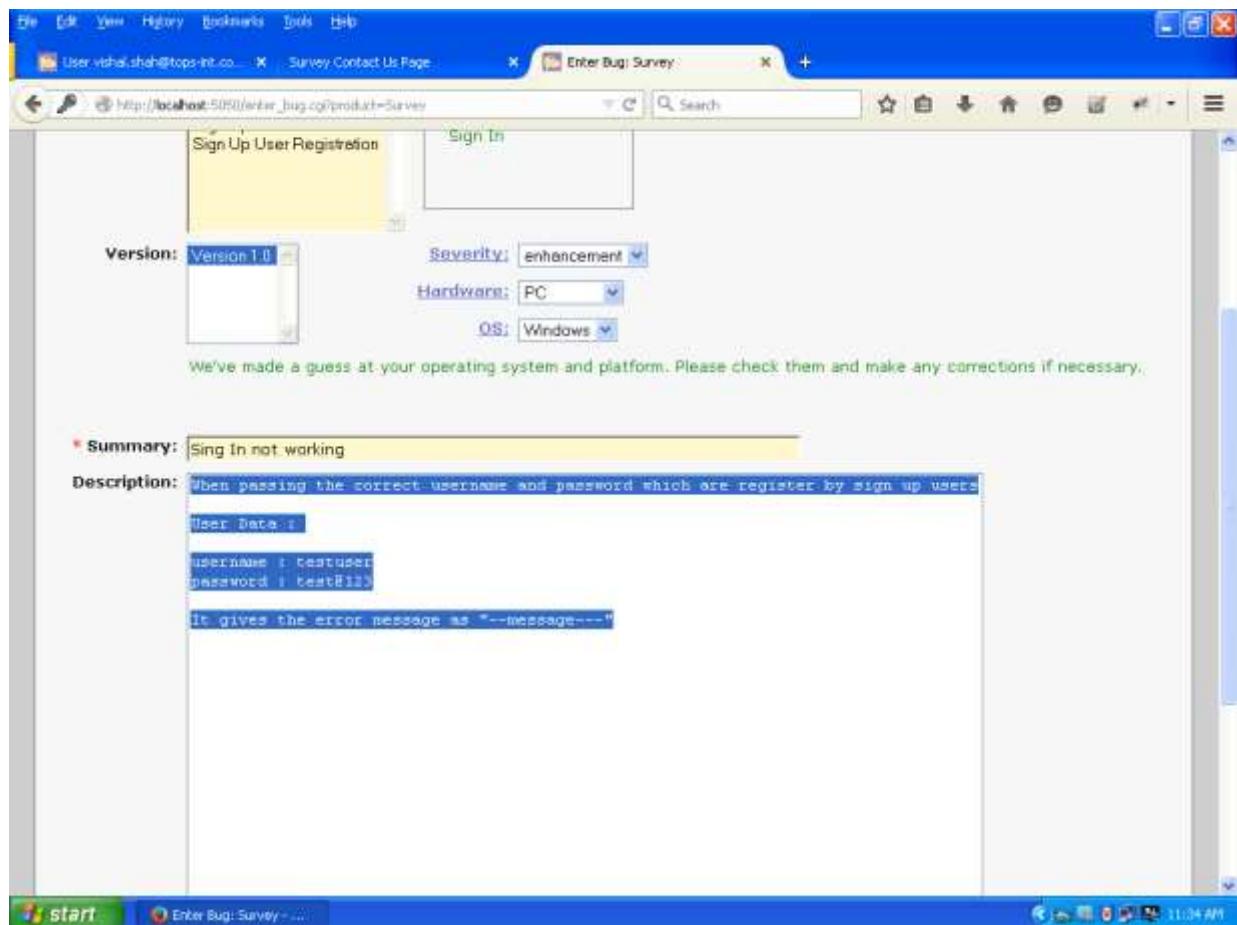
**Version:** Version 1.0      **Severity:** enhancement  
**Hardware:** PC      **OS:** Windows

We've made a guess at your operating system and platform. Please check them and make any corrections if necessary.

**\* Summary:** [Text input field]

**Description:** [Large text area]

# Add details into the new bug entry



# Bugs which were entered

The screenshot shows a Mozilla Firefox browser window with the following details:

- Title Bar:** File Edit View History Bookmarks Tools Help
- Address Bar:** Survey Contact Us Page > Bug List
- Page Title:** Bugzilla - Bug List
- Page Content:**
  - Date: Fri Nov 27 2015 11:37:32 AFT
  - Message: Bugzilla would like to put a random quote here, but no one has entered any.
  - Product: Survey Component: Sign In Resolution: --
  - Table Headers: ID ▲, Rev ▲, Pri ▲, OS ▲, Assignee ▲, Status ▲, Resolution, Summary
  - Table Data:

ID	Rev	Pri	OS	Assignee	Status	Resolution	Summary
1	cn	---	Wind	tops@tops-int.com	NEW	---	Sign In not working
  - Message: One bug found.
  - Buttons: Long Format, XML, CSV, Feed, iCalendar, Change Columns, Edit Search, Remember search as [ ]
  - Text: File a new bug in the "Survey" product.
  - Page Bottom: Home | New | Browse | Search | Search | ? | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com
  - Page Bottom: My Bugs
- Taskbar:** start, Bug List - Mozilla Firefox, Edit values for which..., 11:37 AM

# Assign the Bug

The screenshot shows a web browser window displaying a bug tracking system interface. The title bar reads "Survey Contact Us Page" and "Bug 1 - Sing In not working". The URL in the address bar is "http://localhost:8080/show\_bug.cgi?id=1". The main content area is titled "Bug 1 - Sing In not working (edit)".

Form fields include:

- Status: NEW ([edit](#))
- Product: Survey
- Component: Sign In
- Version: Version 1.0
- Platform: All Windows
- Importance: High critical
- Assigned To: tops ([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](#))
- See Also: [Add Bug URLs:](#)
- URL: <http://survey-javatops.rhcloud.com/contact.jsp>
- Keywords:
- Depends on:
- Blocks:

Below the form is a table for time estimation:

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)

Summary text: Summarize time (including time for bugs blocking this bug)

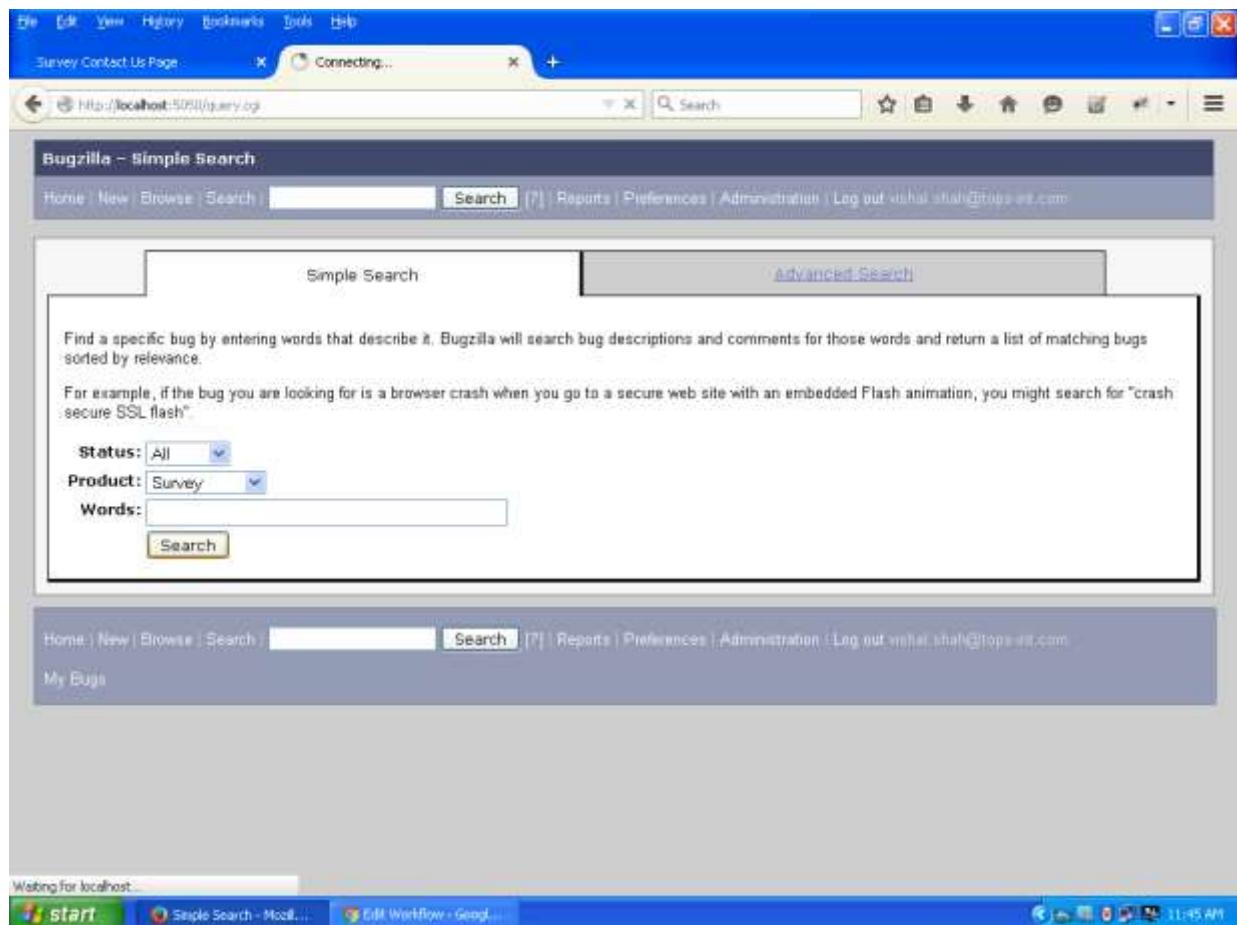
Attachments:

[add an attachment](#) (proposed patch, testcase, etc.)

Additional Comments:

Windows taskbar at the bottom shows the application is running.

# Search the Entered Bug



# Resolved Bug Status

Bugzilla - Bug List

Fri Nov 27 2015 11:45:33 AFT

Bugzilla would like to put a random quote here, but no one has entered any.

Product: Survey

ID	Rev	Pri	OS	Assignee	Status	Resolution	Summary
1	cn	High	Wind	tops@tops-int.com	PESO	FIXE	Sing In not working

One bug found.

Long Format  
XML  
CSV | Feed | iCalendar | Change Columns | Edit Search | Remember search as

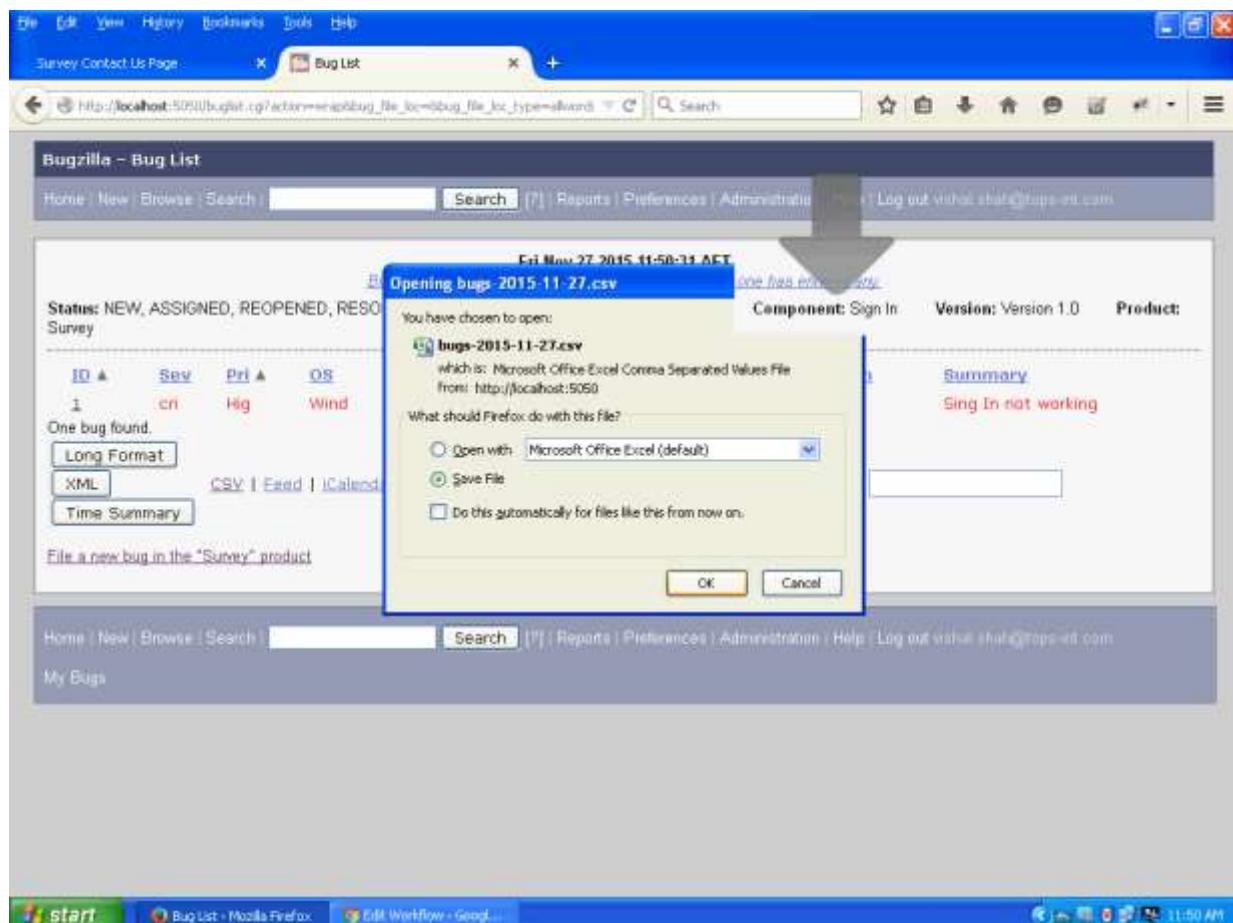
File a new bug in the "Survey" product.

Home | New | Browse | Search | Search | Help | Reports | Preferences | Administration | Help | Log out vishal.shah@tops-int.com

My Bugs

start Bug List - Mozilla Firefox Edit Workflow - Google Chrome 11:45 AM

# Saving the Bug entries in the .csv format



TOPS

# Assigning Privileges to New Group Added

The following changes have been made to the 'Developer' group:

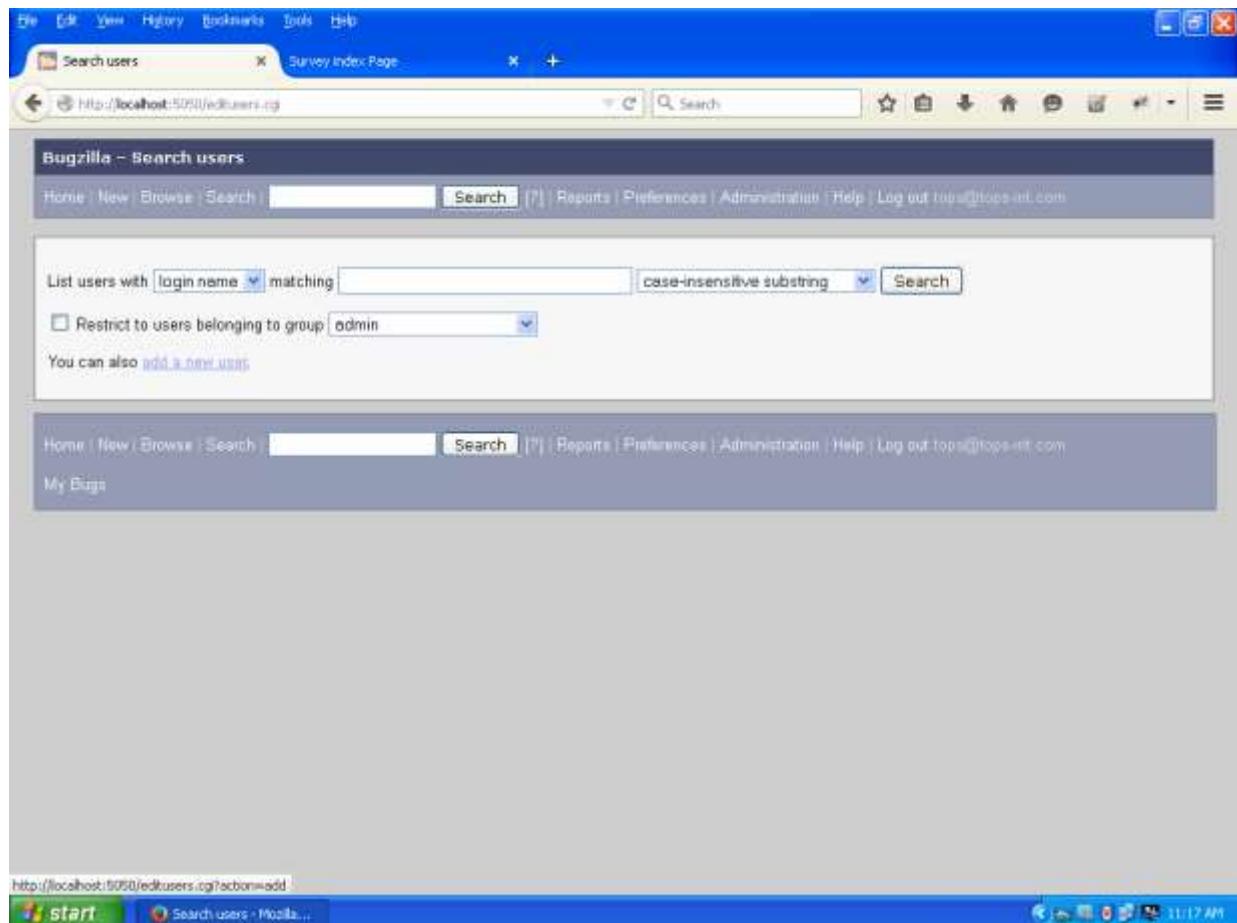
- The following groups are now members of this group: editclassifications
- The following groups are no longer members of this group: editkeywords

<b>Group:</b>	Developer
<b>Description:</b>	Developer Team
<b>User Regexp:</b>	
<b>Icon URL:</b>	
<b>Use For Bugs:</b>	<input checked="" type="checkbox"/>

**Group Permissions**

Groups That Are a Member of This Group ("Users in X are automatically in Developer")		Groups That This Group Is a Member Of ("If you are in Developer, you are automatically also in...")	
Add (select to add)	Current (select to remove)	Add (select to add)	Current (select to remove)
admin bz_canusewhineelthers bz_canusewhines bz_sudoers bz_sudo_protect creategroups editcomponents editkeywords	conconfirm editbugs editclassifications	admin bz_canusewhineelthers bz_canusewhines bz_sudoers bz_sudo_protect creategroups editcomponents editusers tweakparams	conconfirm editbugs editcomponents editkeywords

# Search the Users



TOPS

## Module - 5

# Automation Testing Tools

## Agenda

- Fundamental of Automation Testing
- Automation Framework
- Load Runner Up
- Selenium IDE

## Fundamentals 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.

TOPS Technologies

# 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.

TOPS Technologies

# Which Test Cases to Automate?

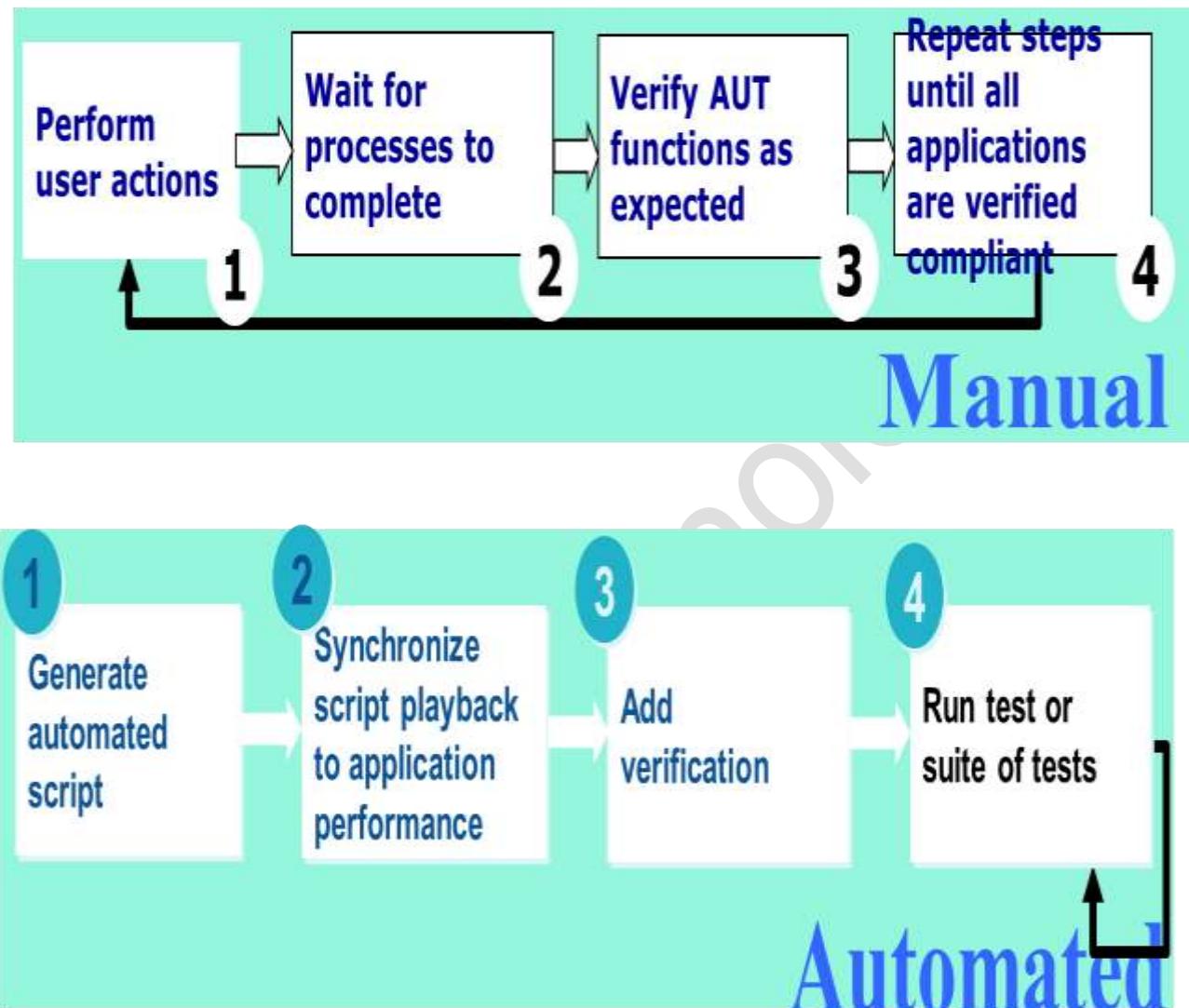
- Test cases to be automated can be selected using the following criterion to increase the automation ROI (Return on Investment)
  - **High Risk - Business Critical test cases**
  - Test cases that are **executed repeatedly**
  - Test Cases that are very **tedious** or difficult to perform manually
  - 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?



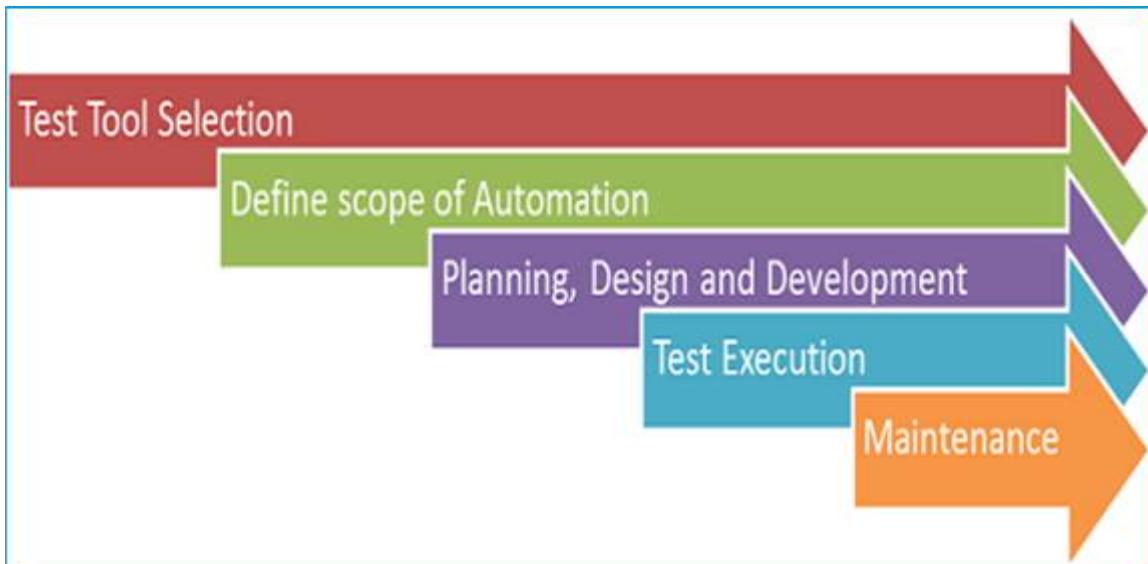
- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• <b>Time consuming</b></li> <li>• <b>Low reliability</b></li> <li>• <b>Human resources</b></li> <li>• <b>Inconsistent</b></li> </ul> | <ul style="list-style-type: none"> <li>• <b>Speed</b></li> <li>• <b>Repeatability</b></li> <li>• <b>Programming capabilities</b></li> <li>• <b>Coverage</b></li> <li>• <b>Reliability</b></li> <li>• <b>Reusability</b></li> </ul> |
|--|--|

# Manual to Automated Testing



# Automation Process

- Test Tool Selection
- Define the Scope of Automation
- Planning, Designing and Development
- Test Estimation
- Maintenance



TOPS Tech

# How to choose Automation Tool?

- Environment Support
- Ease of use
- Testing of Database
- Object identification
- Image Testing
- Error Recovery Testing
- Object Mapping
- Scripting Language Used
- Minimize training cost of selected tools
- Support for various types of test – including functional, test management, mobile, etc...
- Support for multiple testing frameworks
- Easy to debug the automation software scripts
- Ability to recognize objects in any environment
- Extensive test reports and results

## 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.
- 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

## 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  $\geq 8$
  - Response time  $\leq 5$  sec.

# Performance Testing Tools

- Segae Silk Performer
- Rational Team Test
- Mercury Load Runner
- Empirix e-Load/RSW
- Soft Light Tools Loader

## Course Prerequisites

- Windows and Windows applications
- Load Testing concepts

## Course Objectives

- Understand the issues involved in load testing Web applications
- Learn your responsibilities as the LoadRunner expert
- Learn how to perform successful load tests using LoadRunner

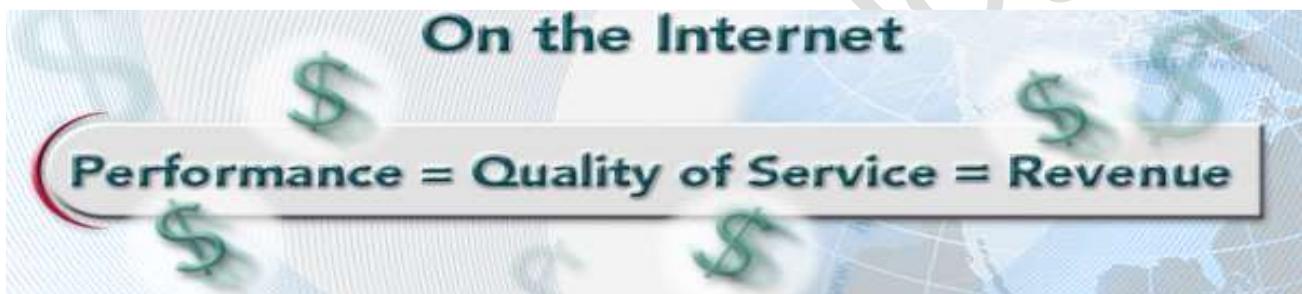
## Session Outline

- Part 1 – Introduction
- Part 2 – Planning
- Part 3 – Using VUGEN
- Part 4 – Controller
- Part 5 – Analysis
- Summary
- References

# Part I – INTRODUCTION

## The Reality

- Fact #1 :
  - Most users click away after 8 second delay
- Fact #2 :
  - \$4.4 billion in revenue lost annually
  - due to poor web performance



Source: Zona Research

...continued

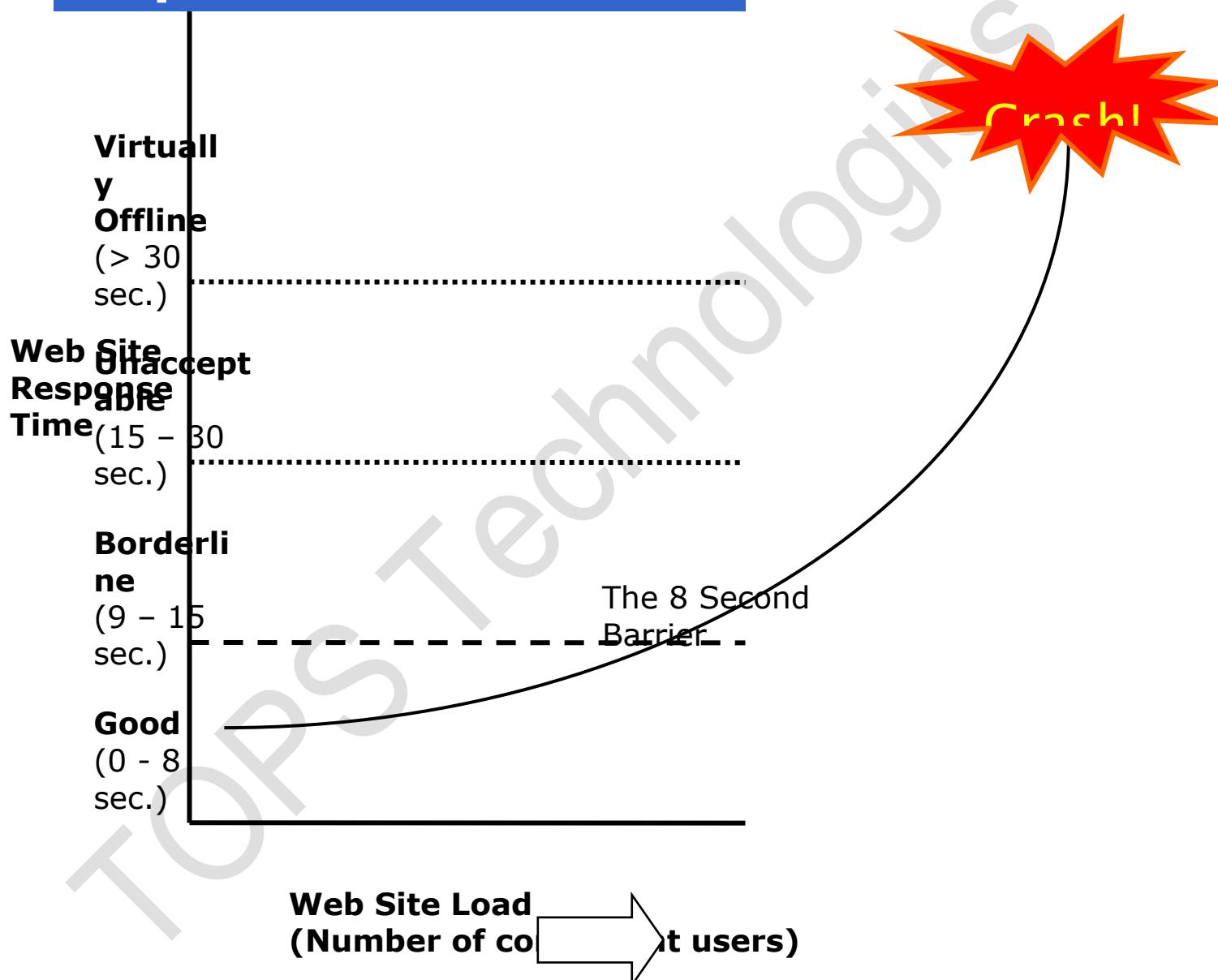
## Why Load Tests?

- The failure of a mission-critical web application can be costly
- Assure performance and functionality under real-world conditions
- Locate and resolve potential problems before it hits on the users.

....continued

# Why Load Tests?

## Performance is Important



**Source: Presentation by Lloyd Taylor, VP Technology & Operations, Keynote Systems**

# Why Load Tests?

- Performance is the key to success of any web based application
- We will get to know the maximum capacity the system can handle for an application
- We can decide whether we should go for Hardware upgrades or Performance tuning

## Objectives of this Course

- Understand the issues involved in load testing Web applications
- Learn your responsibilities as the LoadRunner expert
- Learn how to perform successful load tests using LoadRunner

## Load Testing

- Many concurrent users running the same application to see whether a system handles the load without compromising functionality or performance.

### Test Objectives

- How many concurrent users can the system handle without increase in the expected response time?
- Can the system serve x concurrent users without any errors?

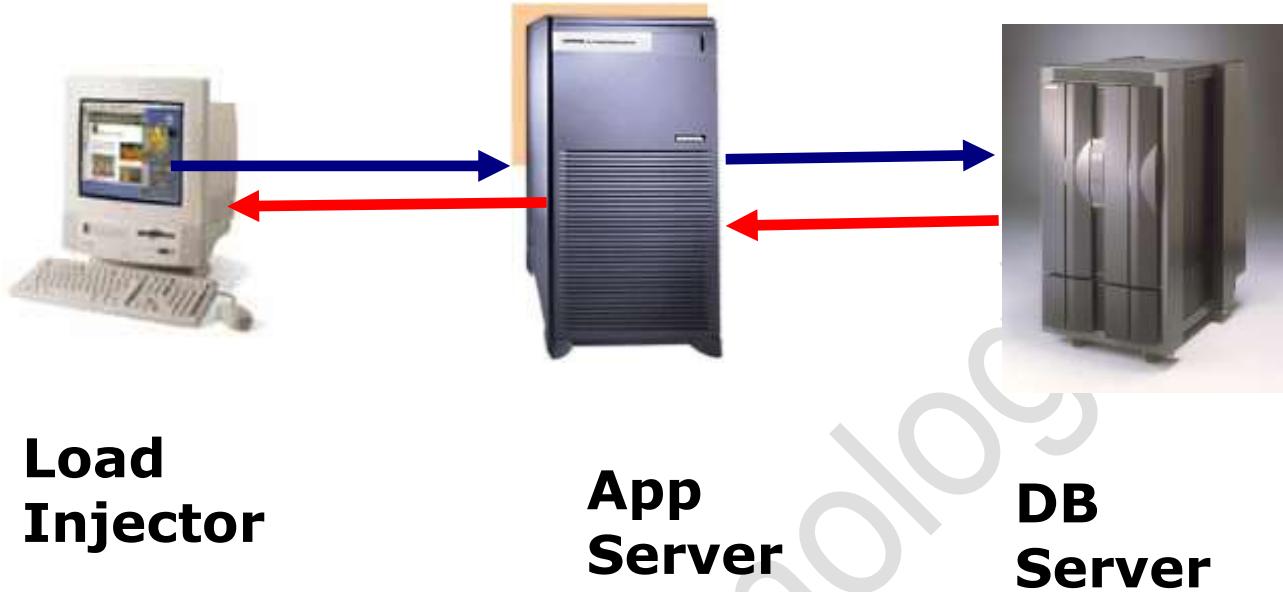
### Important Terminologies

- System
- Response Time
- Think Time
- Work Load
- Transaction Mix
- Throughput

# The System



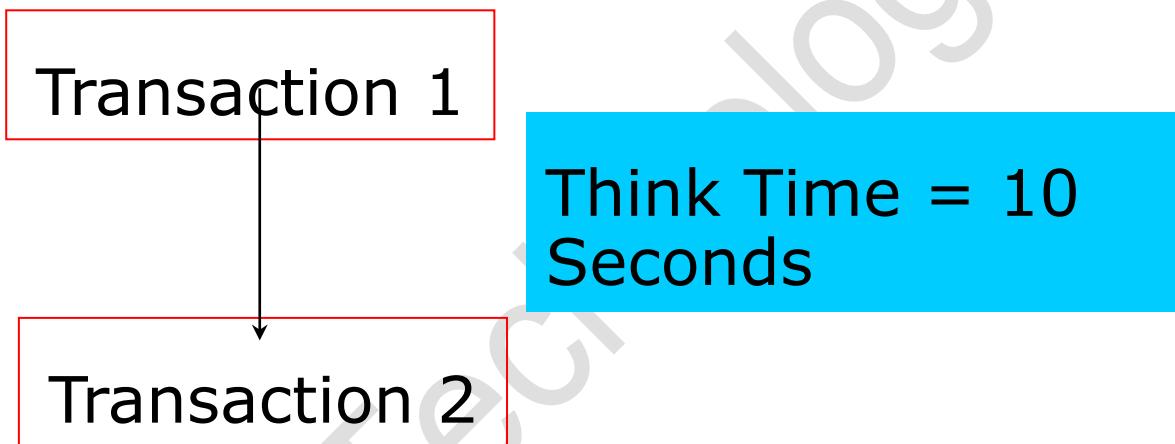
# Response Time



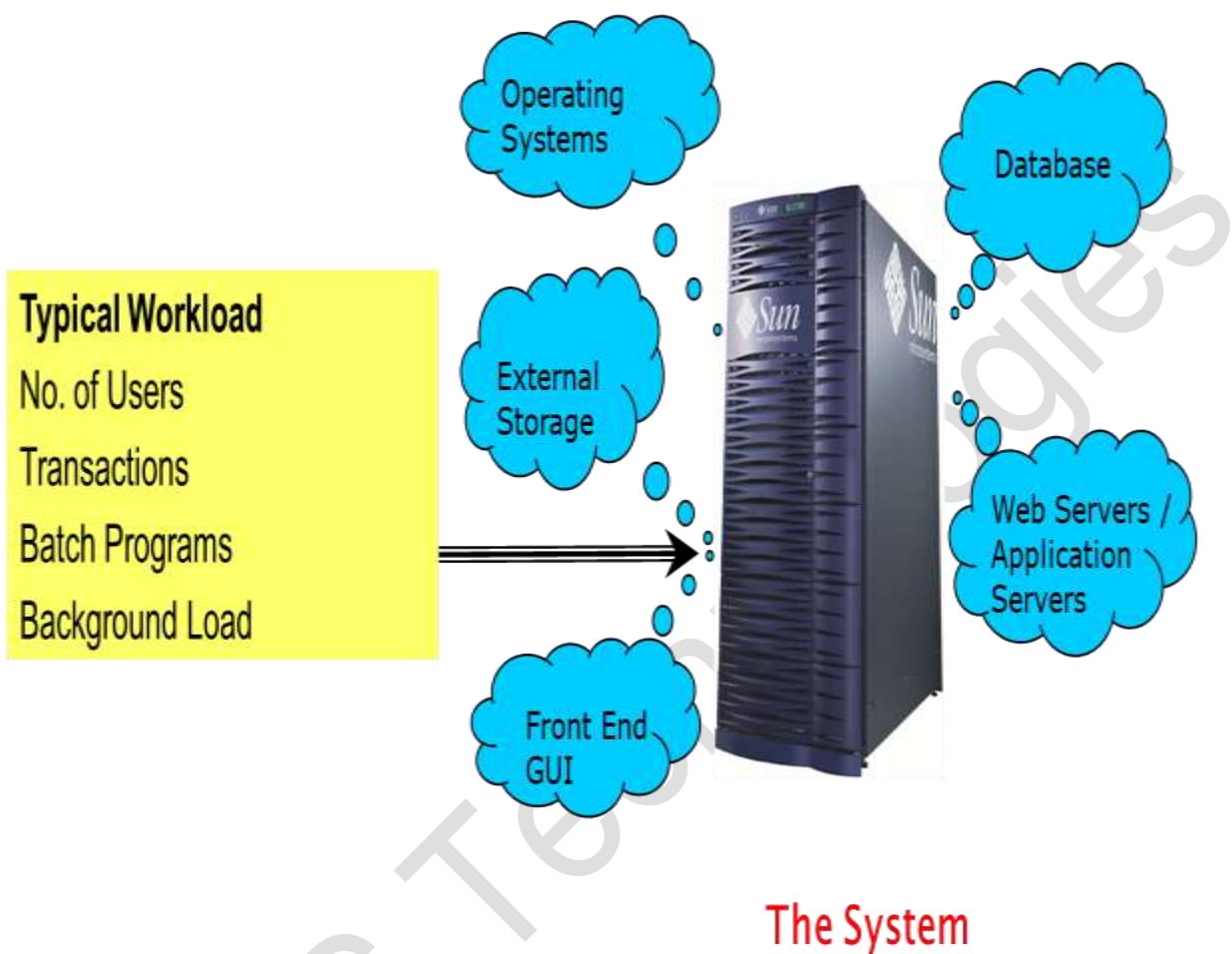
Time in which the system responds for a particular Transaction request

## Think Time

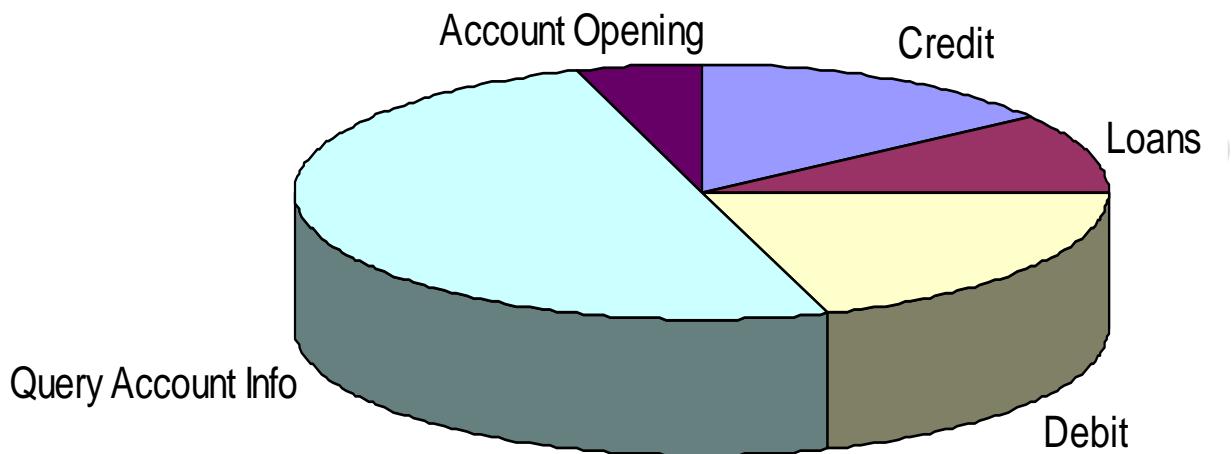
Time taken for selecting a new Transaction after the response for the previous Transaction has been received



# Workload



# Transaction Mix



Varying execution frequency of different Transactions

## Throughput

The amount of work that can be performed by a system or component in a given period of time

- Transactions per Second
- Web Interactions per Second
- Pages per Second
- Bytes per Second

## The Steps Involved in Load Testing

- Planning Load Tests
- Creating Vusers
- Creating Scenarios
- Executing Scenarios
- Analysis of the system under Load

## Part II – PLANNING

### Objectives of this Chapter

- Information that are required for load testing a system
- Where to obtain these information from?
- Organizing the system information collected
- Using the information effectively to carry out Load Tests.

### Business Transactions

A business transaction is a set of user actions performed with an application to accomplish a business task.

Examples:

- Reserving flight tickets
- Buying a computer from a online store
- Logging into your web-based email service

### Vital Information for any Load Test

- The architecture of the system
- A brief idea about the application.
- The list of business transactions
- The time at which the system has peak load
- The acceptable response time for the transactions
- No. of concurrent users the system is expected to handle during peak load

## How to Obtain?

- IT documentation
- Application users
- IT Managers
- System Administrators
- Server and online statistics
- Developers

## Getting Organized

- Select the Transactions to automate
- Decide on the Transaction Mix
- Obtain User details for the transactions
- The tables these transactions query
- The web server / application server components invoked by the transactions

### Select the transactions

- Give priority to business critical transactions
- Give priority to transactions that have high database activity
- Give priority to transactions that have high web server or application server activity
- Know the peak load time
- Impact of the failure of a particular transaction on business
- The transactions that are fired during peak load
- the no. of times they are fired during peak load

### User details

- For each of the chosen transactions find
- Who are the users who fire the transaction?
- How many concurrent users are there?
- How often they fire the transactions?
- For each of the chosen transactions find

### Tables queried



- The tables the transaction queries

- The tables the transaction updates

#### **Server components invoked**

- The servlet / jsp that are invoked by the transaction.

## **Part III - Using VUGEN**

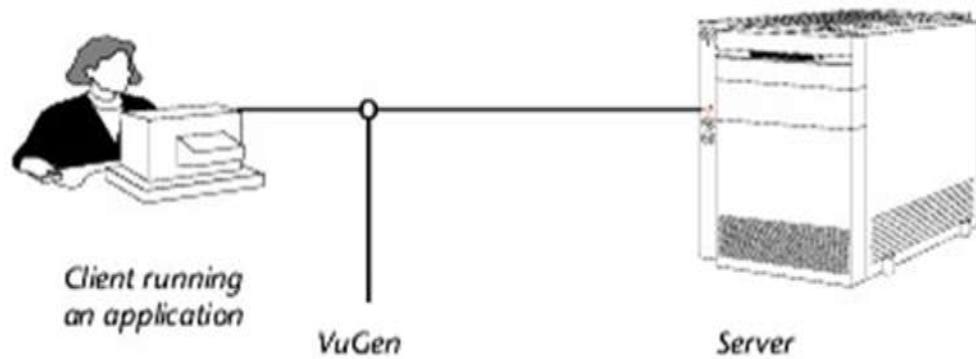
### **Objectives of this Chapter**

- Introduction to Virtual User Technology
- Record scripts
- General options
- Recording options
- Runtime options
- Replay recorded scripts

### **Virtual User Technology**

- Simulates a real user
- Requires less resources – machines and people
- Greater control over test execution
- Can synchronize actions performed by users
- Collect and analyze results in a better way
- VUsers can communicate directly with a server by executing calls to the server API-without relying on client software

## VuGen



- Capture a business transaction by recording user actions performed
- Monitors the communication between the application and the server
- Generates the required function calls
- Inserts the generated function calls into a Vuser script

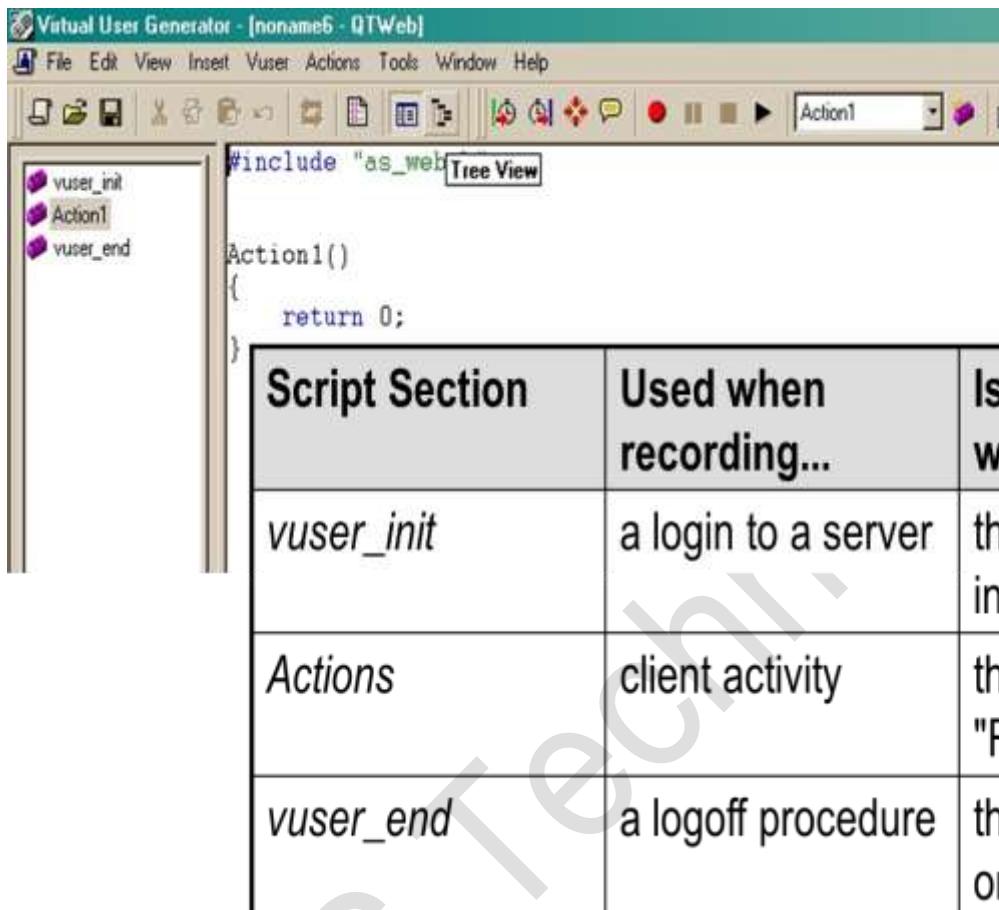
# Recording a Script

## New VUser

- Click on New Vuser
- Select the new Virtual User from the list of protocols
- Click OK
- The required header files are automatically included in the script



# Vuser Script Sections



The screenshot shows the Virtual User Generator software interface. On the left, there is a tree view pane showing nodes for 'vuser\_init', 'Action1', and 'vuser\_end'. The main area displays the following C-like pseudocode:

```
#include "as_web"
Tree View

Action1()
{
    return 0;
}
```

Below this, there is a table summarizing the Vuser Script Sections:

Script Section	Used when recording...	Is executed when...
vuser_init	a login to a server	the Vuser is initialized (loaded)
Actions	client activity	the Vuser is in "Running" status
vuser_end	a logoff procedure	the Vuser finishes or is stopped

# Recording Options

## Recording Mode

This allows us to specify the information to be recorded and which functions to be used when generating a script

- HTML Mode
- URL Mode

## HTML Mode

The HTML mode is based on user actions, and the scripts contain functions that correspond directly to the action taken

### Example:

```
...
web_link("Enterprise Systems Performance",
"Text=Enterprise Systems Performance ",
"Snapshot=t4.inf",
LAST);
```

## URL Mode

The URL mode is based on HTTP requests sent to the server as a result of user actions.

**Example:**

...

```
web_url("Enterprise Systems Performance",
"URL=http://www.tcs.com/esp.html",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://www.tcs.com/atc?... ,
"Snapshot=t4.inf",
"Mode=URL",
LAST);
```

# Deciding on Recording Mode

HTML Mode	URL Mode
Intuitive and easy to understand	Not as intuitive as the HTML scripts
Scalable	More scalable and effective for creating a load test

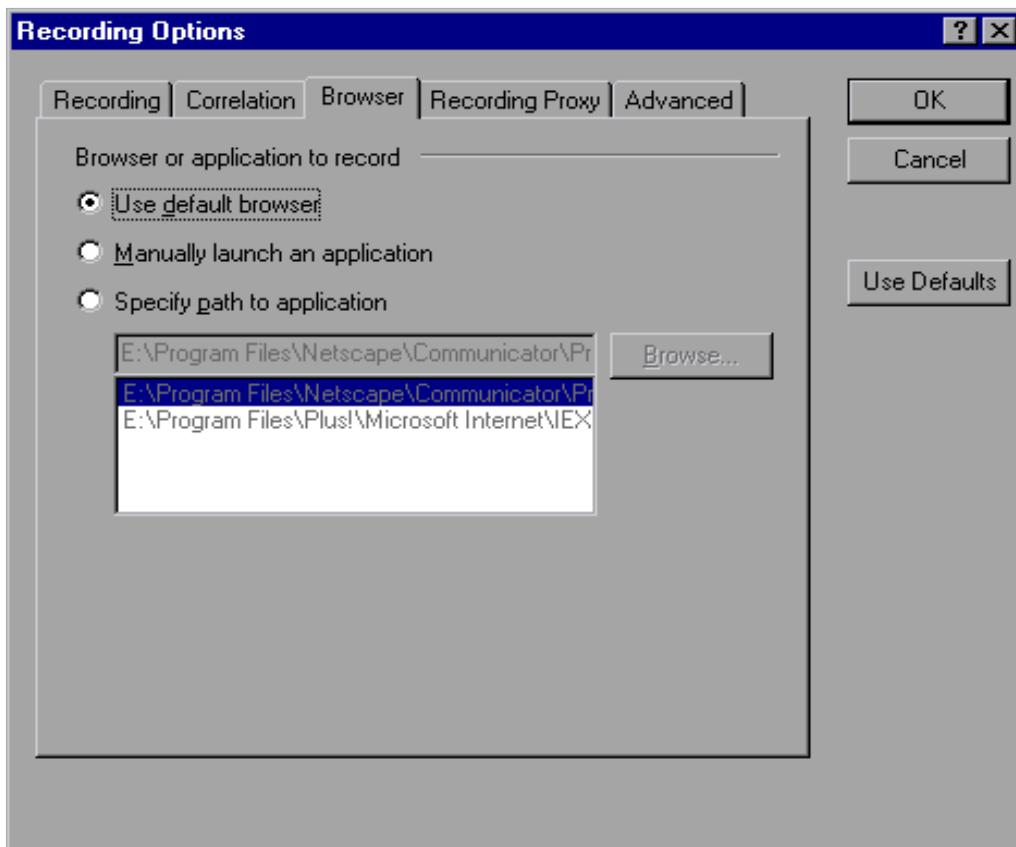
## Deciding on Modes

- For browser applications, use HTML mode.
- For non-browser applications, use URL mode.

Note: The option of mixing recording modes is available for very advanced users for performance tuning

# Other Recording Options

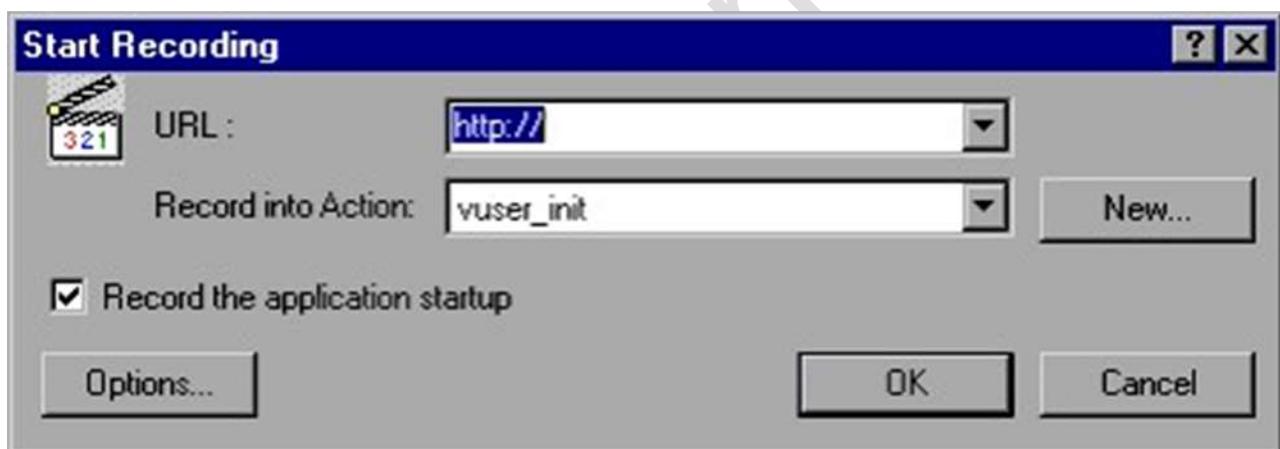
## Selecting the Browser



# Recording Scripts

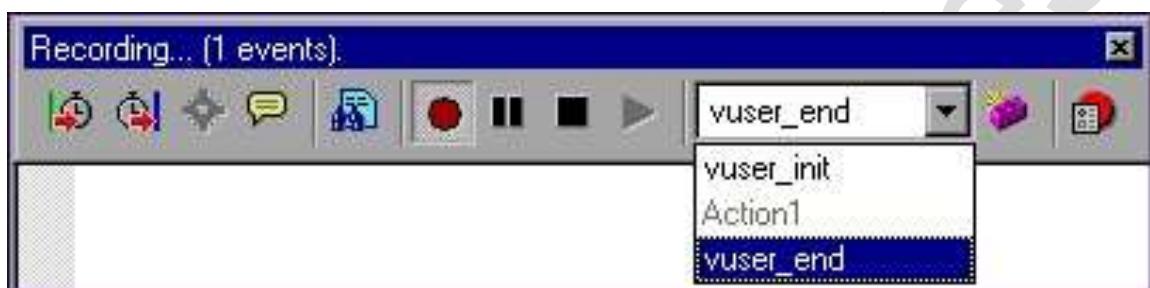


- Click on Record Button (●)
- Record dialog box pops up.



# Recording Scripts

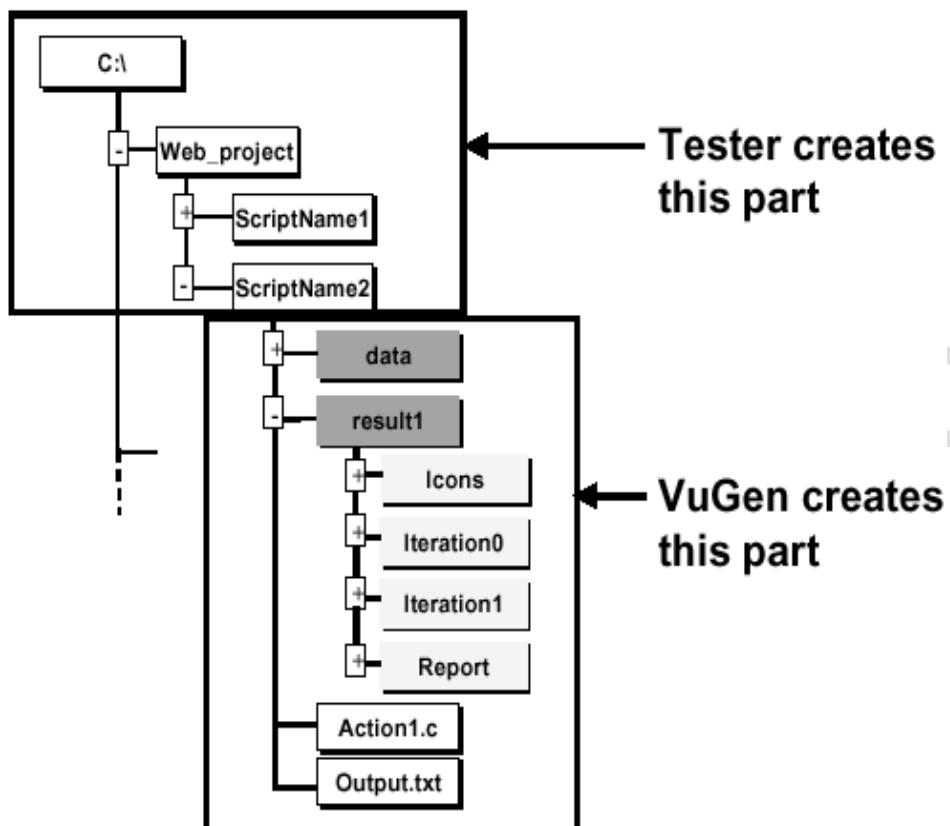
- Click Stop button ( ) to end recording
- Save the script.



## Note:

- Record user transaction in the appropriate sections
- Do not name a script init, run, or end, as these names are used by VuGen.

- While recording, VuGen creates a series of configuration, data, and source code files which contain Vuser run-time and setup information.
- The results of each iteration of a script are stored separately.
- The Iteration0 directory is for system use and should be ignored by the tester.



# Load Runner Transactions

- A LoadRunner transaction measures start to end performance of any user action that is made from the browser.
- Click the Start Transaction button (  ) after selecting the starting point for measurement 
- Click the End Transaction button (  ) after selecting the end point for measurement.
- Select Transaction Status (optional) – LR\_AUTO, LR\_PASS, LR\_FAIL

**Note:** LoadRunner Transactions can also be inserted during recording

# Parameterization

## Parameter

- A parameter is a placeholder which re-places a recorded value in a Vuser script. At run time, a value from an external source is substituted for the parameter.

## Parameterize Input data

- During a run, VuGen replaces each parameter with a value from a data file or another external source, and inputs it to the Web application

# Why Parameterization?

Parameterization solves certain problems that may occur during playback

- Date constraints make recorded date invalid
- Unique constraints prevent reusing recorded data
- Data caching
  - Response times are inaccurate
  - Real user activity is not emulated
  - Servers are not exercised

## Parameterization

### Unique Constraint

#### Problem

- Some fields require a unique ID as in the case of a Primary Key or fields in database with Unique constraint
- How to measure the performance of the process which creates a new product?

#### Solution

- Parameterize the unique id fields
- After running, delete the test ID numbers from the system so the Vuser can be run again

# Parameterization

Data is Cached

Problem

- Vusers all use the same data
- Data caching makes load test response times shorter than when real users load the system

## Solution

- Parameterize input data to force the system to get data from a database

How to create data pool?

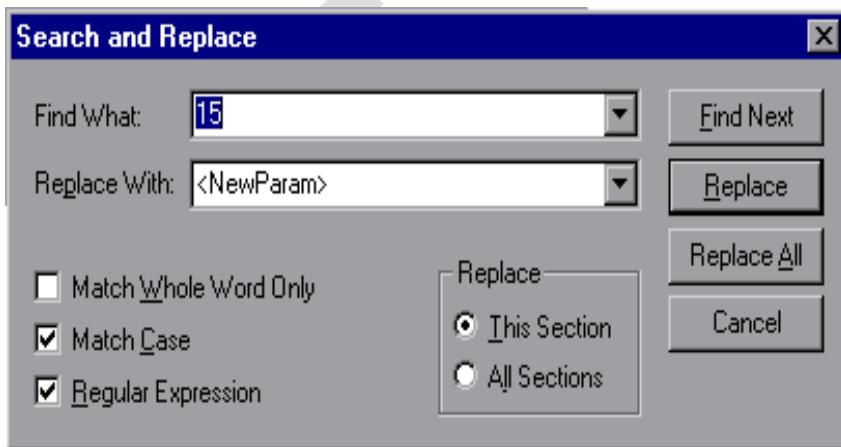
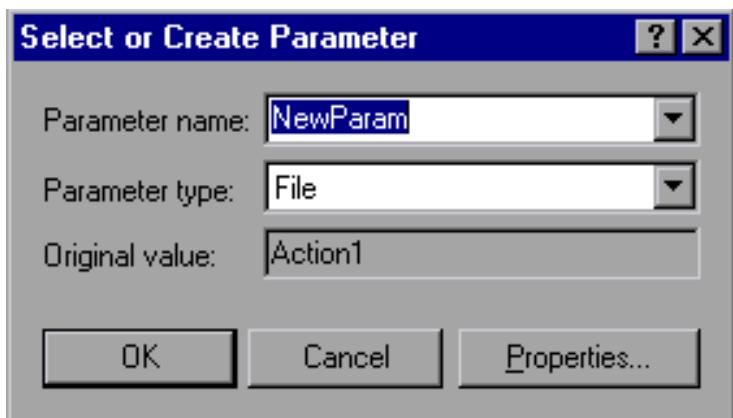
- Extract master data from database
  - Write queries on tables being accessed by the business transactions.
  - Spool data to a file
  - Write AWK scripts to generate a proper data pool from the spooled file
- Create valid transactional data
  - Predict according to type of data
  - Obtain from a functional expert

The Process

- Determine which fields to parameterize
- Replace recorded values with parameters
- Decide which parameter type to use
- Choose the data access method and the number of iterations
- Run the Vuser and analyze the results to verify correct execution

# Parameterization – The Process

- Right click on the data to parameterize
- select “Replace with a Parameter”
- The Select or Create Parameter dialog box opens.
- Type a suitable Parameter Name
- Select a Parameter Type
- Right-click the parameter name and select “Replace More Occurrences” to replace the data by the parameter name.



# Correlations

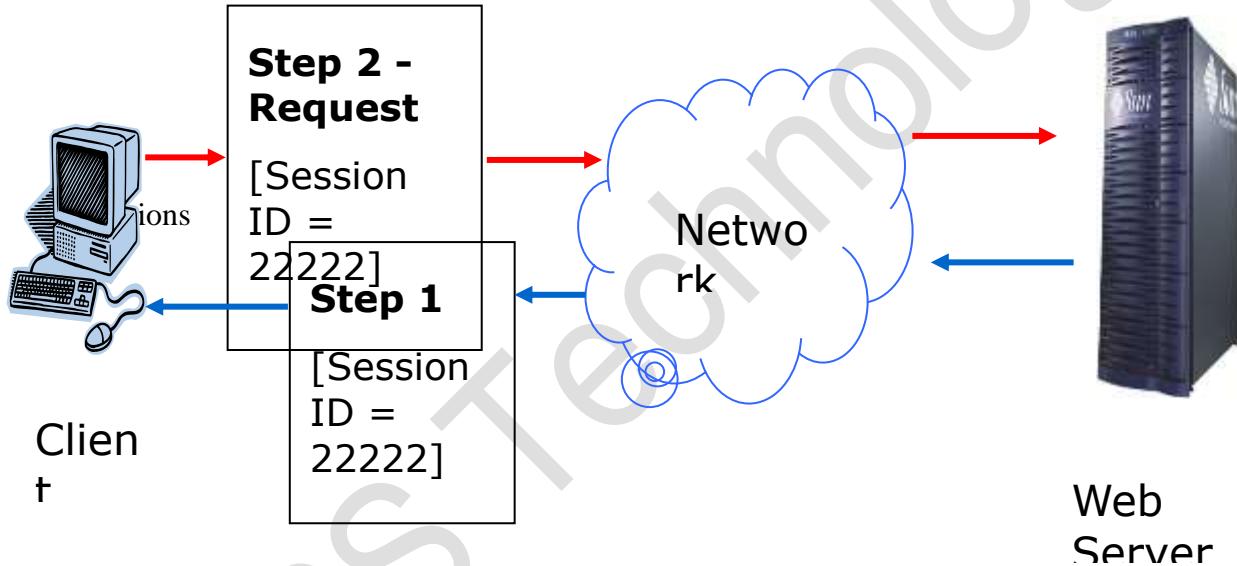
- In applications output of the previous transaction is used as an input for the next transaction.
- For example:
  - System generated session ids
  - URLs that change each time you access the Web page
  - Fields (sometimes hidden) recorded during a form submission
- The data is only good for the current session

## Correlate the data

- Capture output value from one step
- Use captured value as input to another step

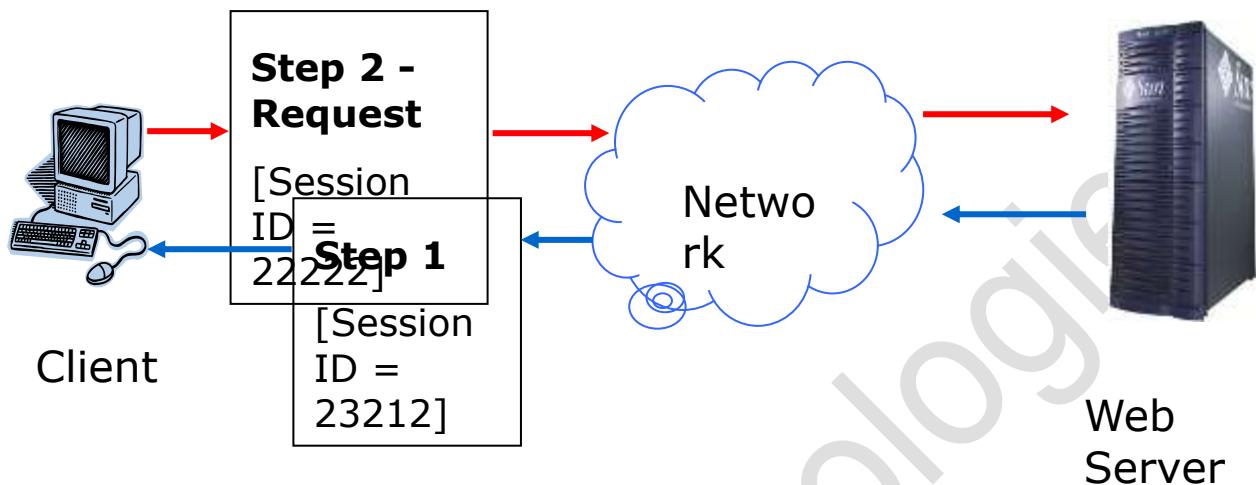
# Correlations

Correlated data is data which is sent to the client from the server, and later sent back to the server by the client



Dynamic data is correlated automatically by the client application (running in the browser) while recording a script

# Correlations



- Reusing recorded dynamic data will cause errors when you replay the script
- Recorded dynamic data must be recorrelated using parameters

## Manual Correlation

- Determine the value to capture
- Find the right and left text boundaries of the value to capture
- Find which occurrence of the text boundaries should be used
- Add a web\_reg\_save\_param function to the script, above the step which requests the page with the value to capture
- Add the parameter name, left boundary, right boundary, and occurrence to the function
- Parameterize the dynamic value in the script every time it occurs
- Verify correct execution

# Automatic Correlation

- There are three types of automatic recording solutions
- Auto-Detect Correlation
- Rule-Based Correlation
- Correlating All Statements

Rule Name	When to Use
Auto-detect Correlation	Detect and correlate dynamic data for supported application servers
Rule-Based	When working with a non-supported application server whose context is known.
Correlate All	Blindly correlate all dynamic data.

# Automatic Correlation

## Auto-Detect Correlation

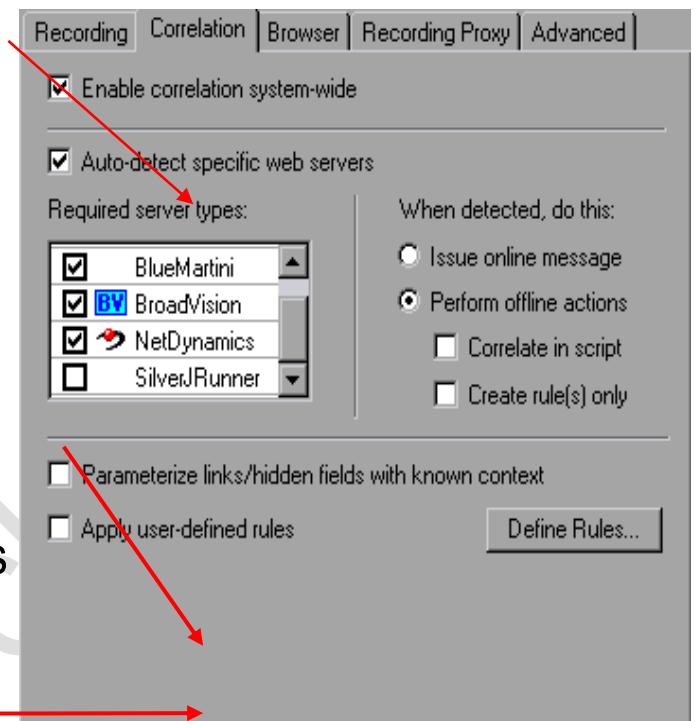
Automatically correlate the recorded statements, for application servers whose contexts

## Correlating All Statements

VuGen correlates all statements

## Rule-Based Correlation

Rule-based correlation requires correlation rules to be defined before recording a session



# Run-Time Settings

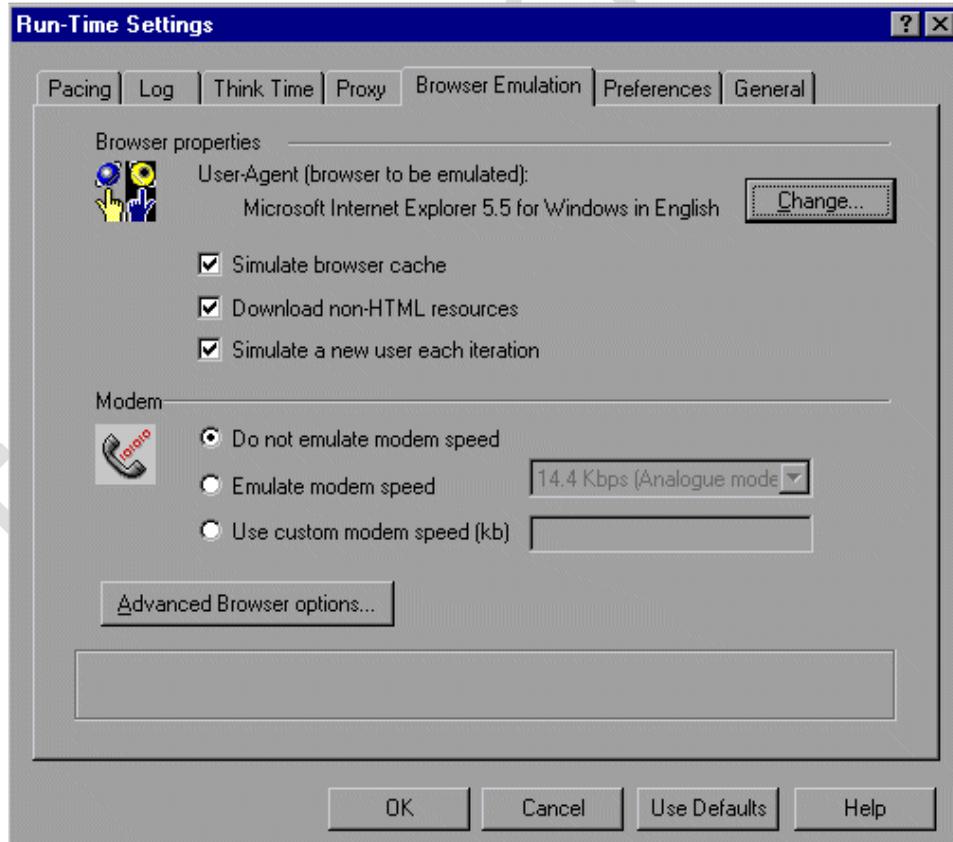
- Control the behavior of the user when running the script
- Configure Web Vusers to emulate real users accessing the Web site

## Important Run-Time Settings

- Browser Emulation
- Think Time
- Logs

### Browser Emulation

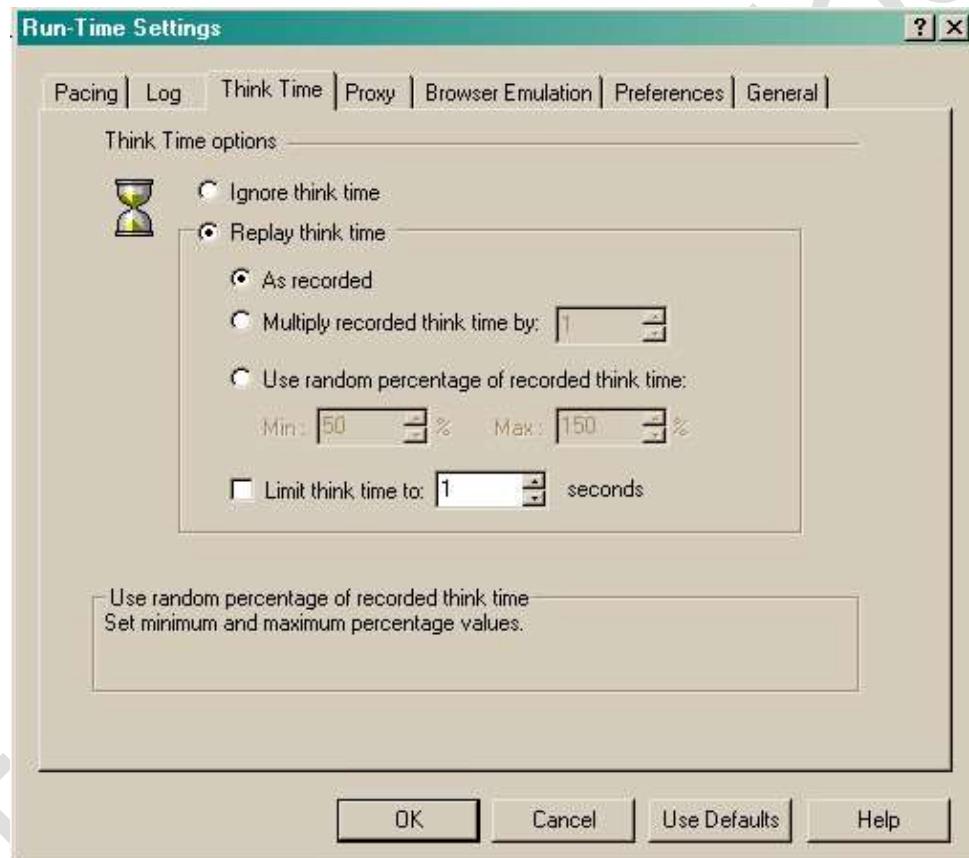
- Specify Browser type
- Simulate cache
- Emulate network capacity
- Emulate modem speed



# Run-Time Settings

## Think Time Settings

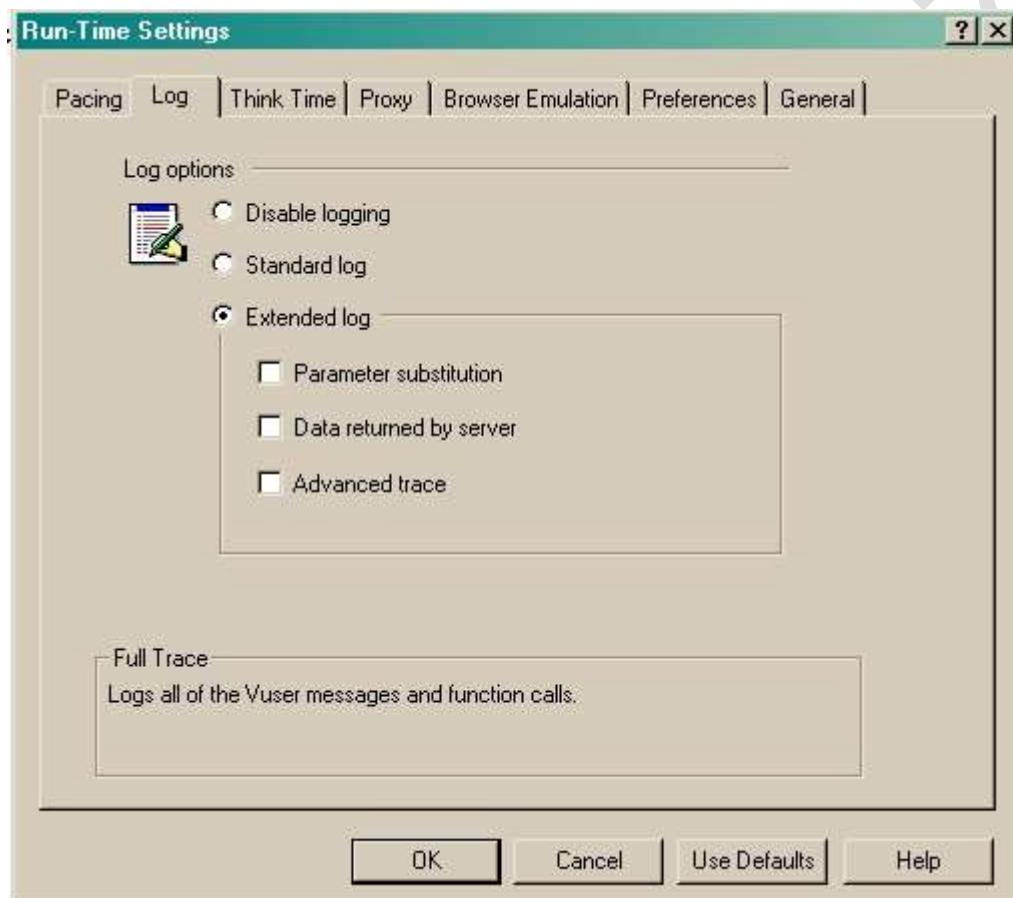
- Specified in seconds
- Ignore recorded think time
- Replay the recorded think time
- Randomize think time
- Multiply the recorded think time by specific values
- Limit the think time to a maximum value



# Run-Time Settings

## Log Settings

- Can disable logging
- Log a specific set of function calls and messages – Standard Log
- Log parameter substitutions
- Log response data from server
- All the messages and function calls raised



## Part IV – Controller

### Objectives of this Chapter

- What is a scenario?
- The different types of scenarios
- How to create and run a manual scenario
- Scheduling a Scenario
- Running a Scenario

### What is a Scenario?

- It describes the events that occur during a testing session
- It is the means by which you emulate a real-life user
- Scenarios include
  - a list of machines on which Vusers run
  - a list of scripts that the Vusers run
  - a specified number of Vusers or
  - Vuser groups that run during the scenario
- Scenarios are created using the LoadRunner Controller

### Types of Scenario

- Manual Scenario

Create a scenario by defining Vuser groups to which are assigned a quantity of individual Vusers, Vuser scripts, and load generators to run the scripts.

- Goal Oriented Scenario

For Web tests, the goals of the tests are defined by us and LoadRunner automatically builds a scenario based on these goals.

# Manual Scenario

## Creating a Manual Scenario

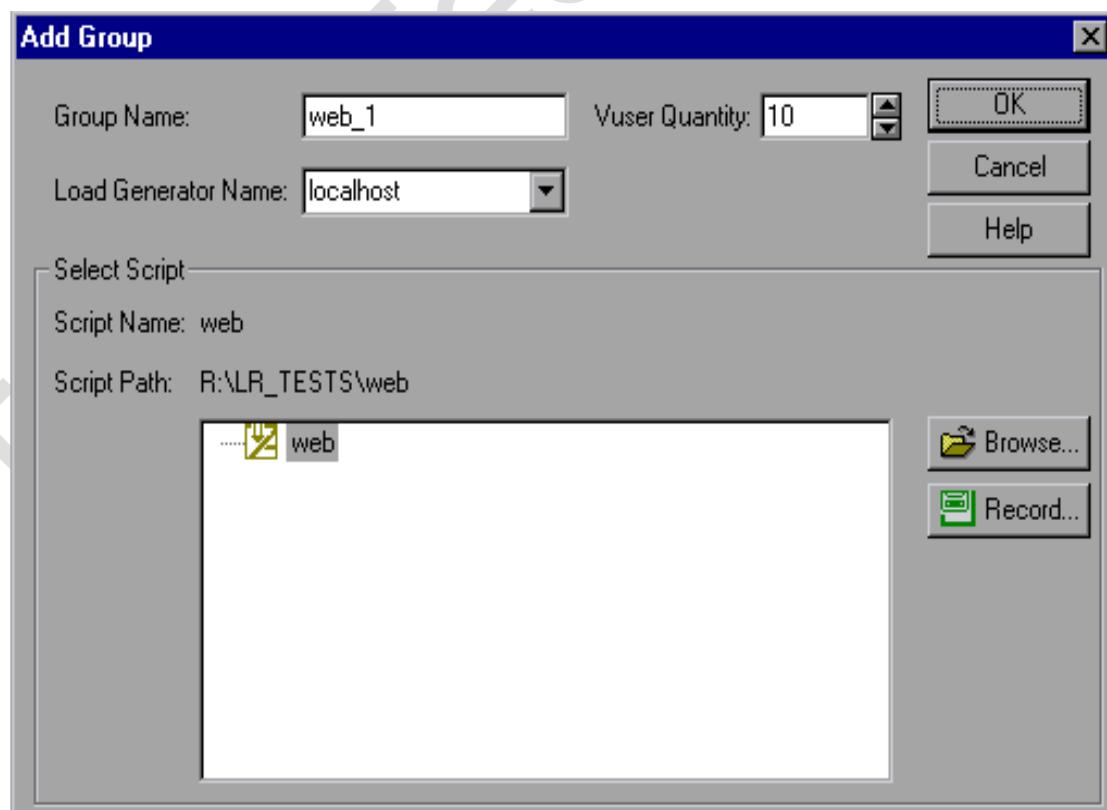
- Create Vuser groups
- Each script selected in the New Scenario dialog box is assigned to a Vuser group
- Each Vuser group is assigned a number of virtual users
- All Vusers in a group can be assigned to run the same script on the same load generator machine

OR

- Different scripts and load generators can be assigned to the various Vusers in a group

## Creating VUser groups

- Click the Add Group button on the right of the Scenario groups window. The Add Group dialog box opens.
- Enter a name for the Vuser group
- Specify the number of Vusers in the group
- 

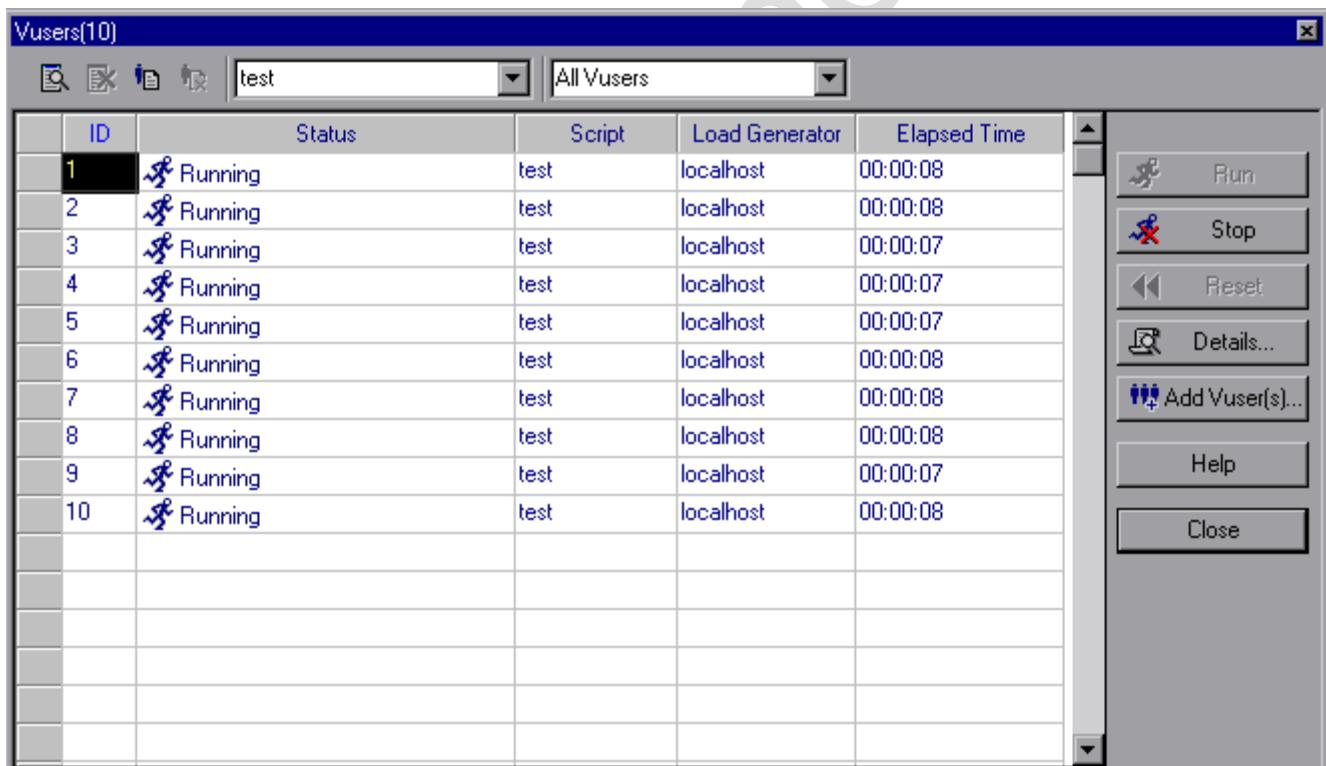


# Creating VUser groups

- Select a load generator from the Load Generator Name list
- Select Add from the Load Generator Name list if the load required generator is not listed.
- Select a script from the script list
- Click OK to close the Add Group dialog box. The new group's properties appear in the Scenario Groups window

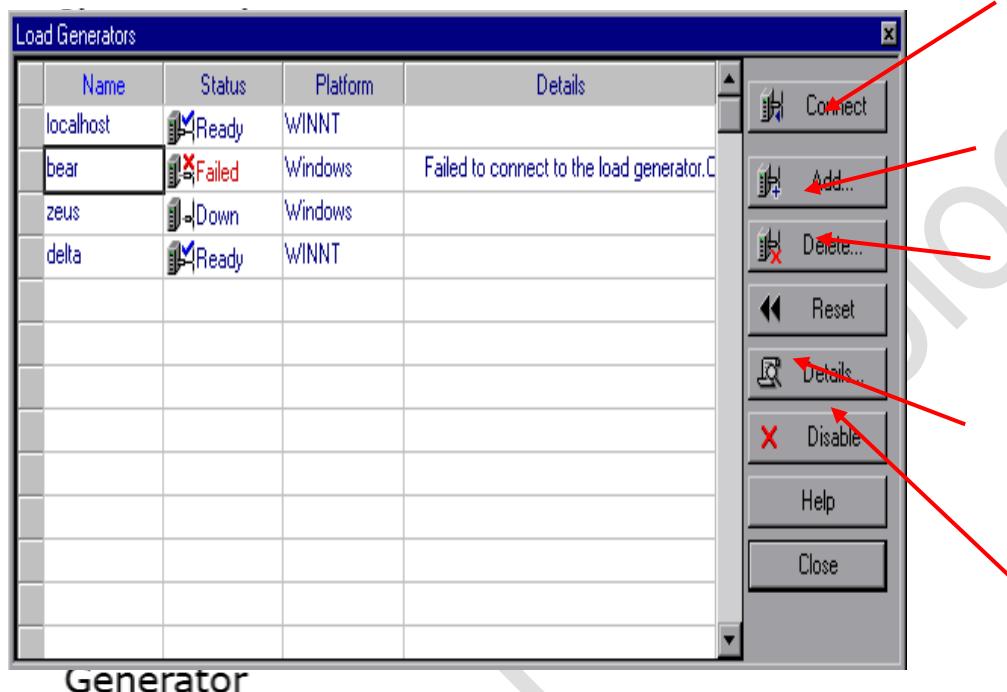
# Configuring VUsers in Vuser Groups

- Select the Vuser group whose Vusers are to be modified, and click the Vusers button on the right of the Scenario Groups window. The Vusers dialog box opens.
- Can change the Script
- Can change the Load Generator
- Can Add Vusers to the group



# Configuring Load Generators

- Click the Generators button to open the Load Generator dialog box.
- Name, Status, Platform and Details of the Load Generators are displayed



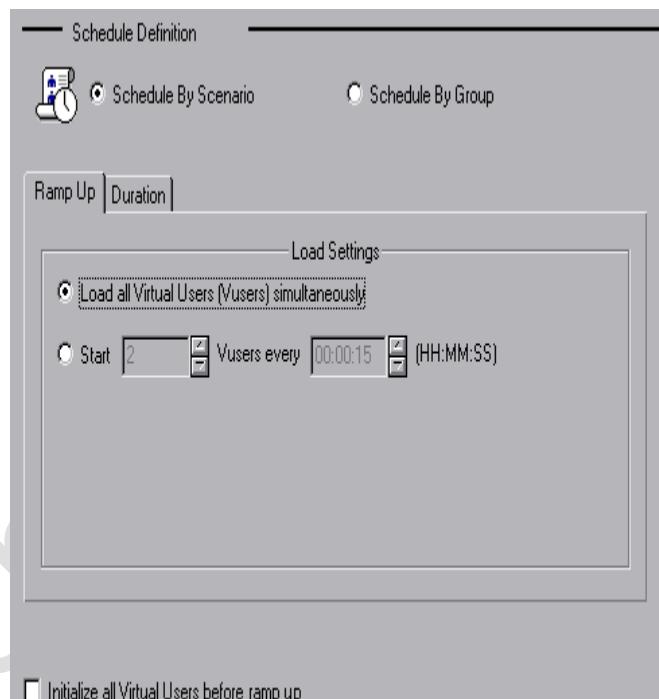
# Configuring Scripts

- Select the Vuser group whose script needs to be modified, and click the Details button on the right of the Scenario Groups window
- Run-Time Settings from the Controller shows the setting of VuGen. If these settings are modified from the Controller, LoadRunner runs the script using the modified settings
- Edit scripts by clicking on View Scripts

# Scheduling a Scenario

- Can instruct LoadRunner to execute a scenario or Vuser group with a delay.
- The time duration of a scenario or Vuser group can be limited. The scenario or Vuser group stops running after the elapse of this time.
- Can stipulate how many Vusers LoadRunner runs within a certain time frame during a scenario or Vuser group.

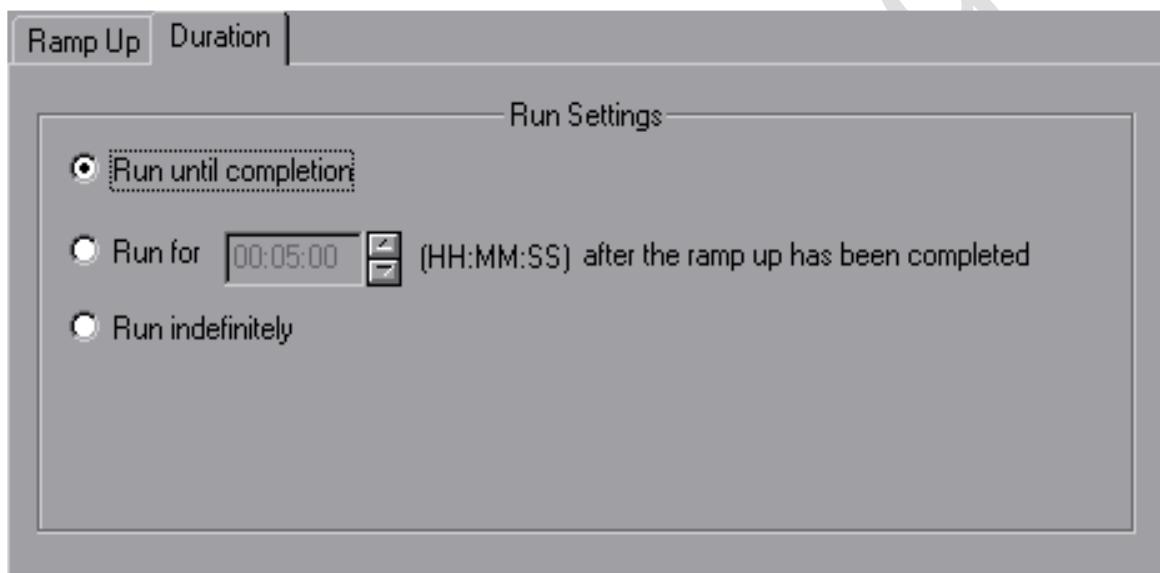
- Select the Scenario Scheduling option
- In the Ramp Up tab:
  - Selecting Load all Virtual Users (Vusers) simultaneously will run all the Vusers at once.



- To gradually run the Vusers, select the number of Vusers to begin running concurrently and the amount of time LoadRunner has to wait between Vuser ramp ups.

# Scheduling a Scenario

- To set the duration of the scenario, click the Duration tab
- There are 3 choices
  - Run until completion
  - Specify the amount of time for which the scenario will run, once all the Vusers have been ramped up
  - Run indefinitely
- Selecting Initialize all Vusers before Ramp-Up, instructs LoadRunner to initialize Vusers before beginning to load them



# Scheduling a Vuser group

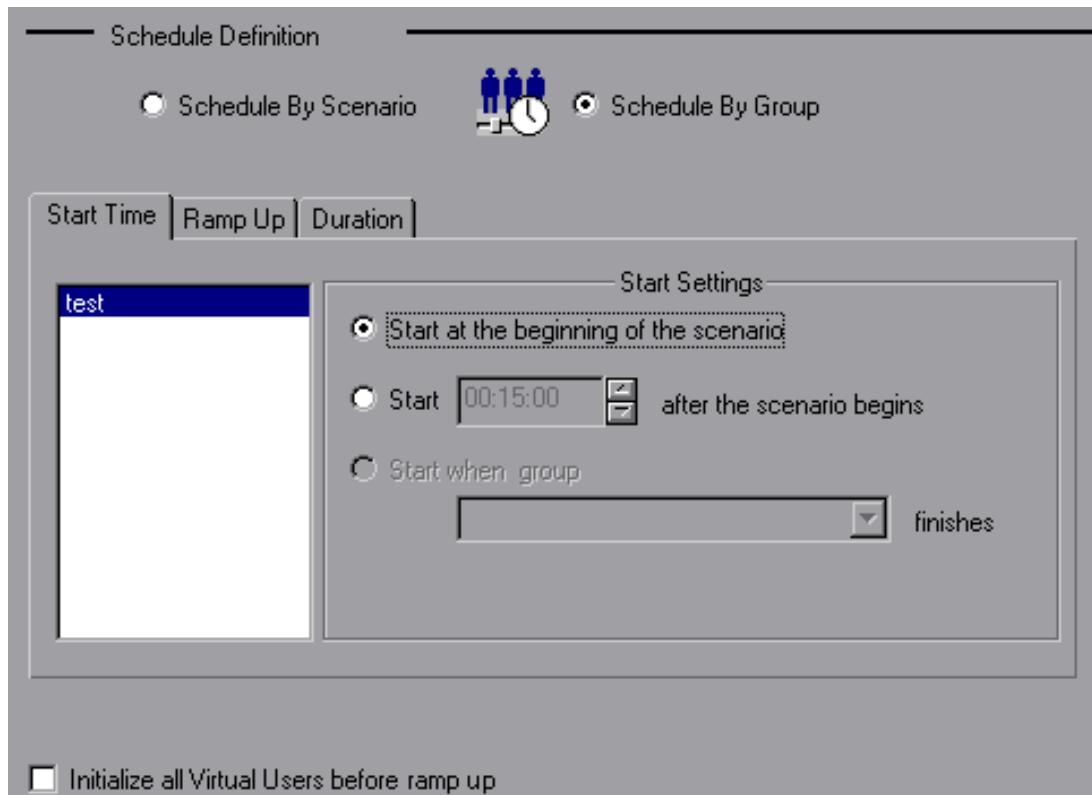
After creating a Vuser group, the group's script execution settings can be scheduled

## Vuser group Execution Settings

- The amount of time after the start of the scenario that the group must wait before it starts running
- The number of Vusers that will run within a specified period of time
- The amount of time the group will run

# Scheduling a Vuser group

- Select Schedule By Group to set the schedule for the Vuser group.
- The options are similar to that of scheduling the scenario



# Running a Scenario

- Two types of execution:
  - Can run all the Vusers and Vuser groups in a scenario OR
  - Can run the specific Vuser groups and Vusers
- When running the entire scenario, LoadRunner does not begin running Vusers until all of them have reached the ready state
- If individual groups or Vusers are run, LoadRunner runs the Vusers as soon as they reach the ready state
- Clicking the Start Scenario in the RUN tab of the Controller starts the execution of the Scenario
- Can Initialize, Stop, Pause and Reset Vuser groups

## Monitoring a Scenario

- We can monitor scenario execution using the LoadRunner monitors.
- Some of the monitors are
  - Run-time
  - Transaction
  - System resource
  - Web resource
  - Web server resource
  - Web application server resource
  - Database server resource
  - Network delay

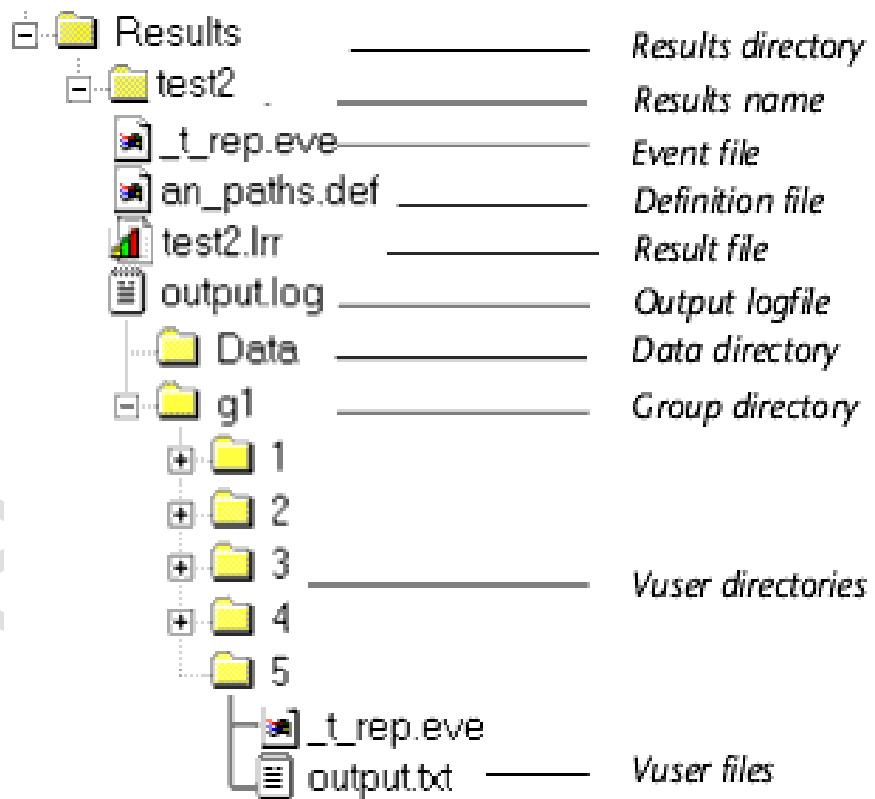
## Part V – Analysis

### Objectives of this Chapter

- Understanding the results directory structure
- Introduction to the Analysis tool
- Creating Analysis graphs
- Interpreting Analysis graphs

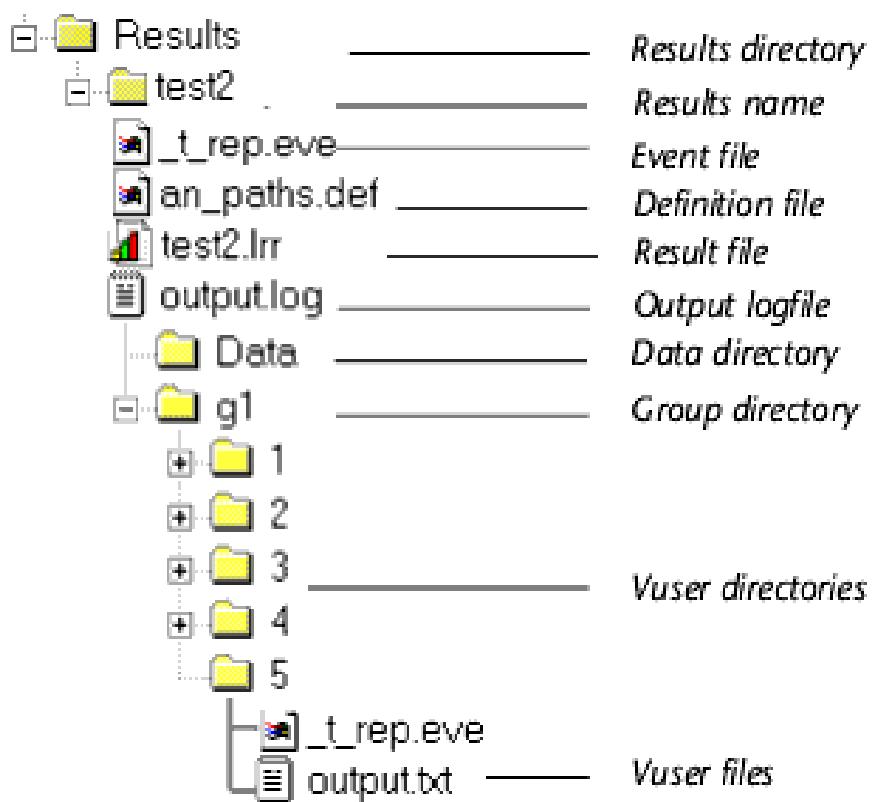
# Results Directory Structure

- In results directory, LoadRunner creates a subdirectory using the results name specified by us
- All the data LoadRunner gathers is placed in that directory
- Every set of results contains general information about the scenario in a result file (.lrr) and an event (.eve) file.



# Results Directory Structure

- During scenario execution, LoadRunner also gathers data from each Vuser and stores it in an event file \_t\_rep.eve and an output file output.txt
- LoadRunner creates a directory for each group in the scenario and a subdirectory for each



Vuser

# The Analysis

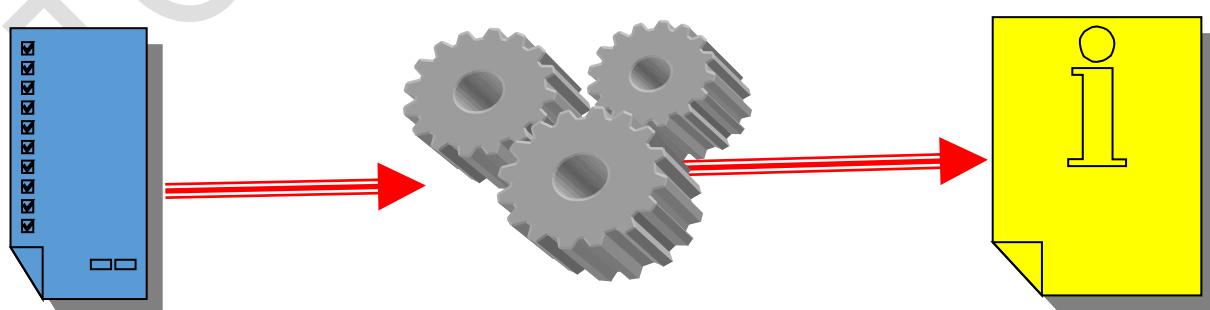
The Analysis graphs helps to determine system performance and provides information about transactions and Vusers by combining results from several scenarios or merging several graphs into one.

- The Analysis is the utility that processes the gathered result information and generates graphs and reports
- The result file with an .lrr extension is the input to the Analysis tool
- An Analysis session contains at least one set of scenario results (lrr file)

## Starting the Analysis

The Analysis can be started

- as an independent application
- or
- directly from the Controller



**Load  
Runner  
results  
file (.lrr)**  
TOPS  
Technologies

**LoadRunne  
r Analysis  
Tool**

**Load Runner  
analysis file  
(.lra)**

# Analysis Graphs

The Analysis graphs are divided into the following categories:

Vusers	Firewalls
Errors	Web Server Resources
Transactions	Web Application Server Resources
Web Resources	Database Server Resources
Web Page Breakdown	Streaming Media
User-Defined Data Points	ERP Server Resources
System Resources	
Network Monitor	

## Analysis graphs

### Vuser

Vuser graphs display information about Vuser states and other Vuser statistics

### Error

Error graphs provide information about the errors that occurred during the scenario execution

### Transaction

Transaction graphs and reports provide information about transaction

performance and response time

TOPS Technologies

## Web Resource

These graphs provide information about the throughput, hits per second, HTTP responses per second, and downloaded pages per second for Web Vusers.

## System Resource

These graphs show statistics relating to the system resources that were monitored during the scenario using the online monitor.

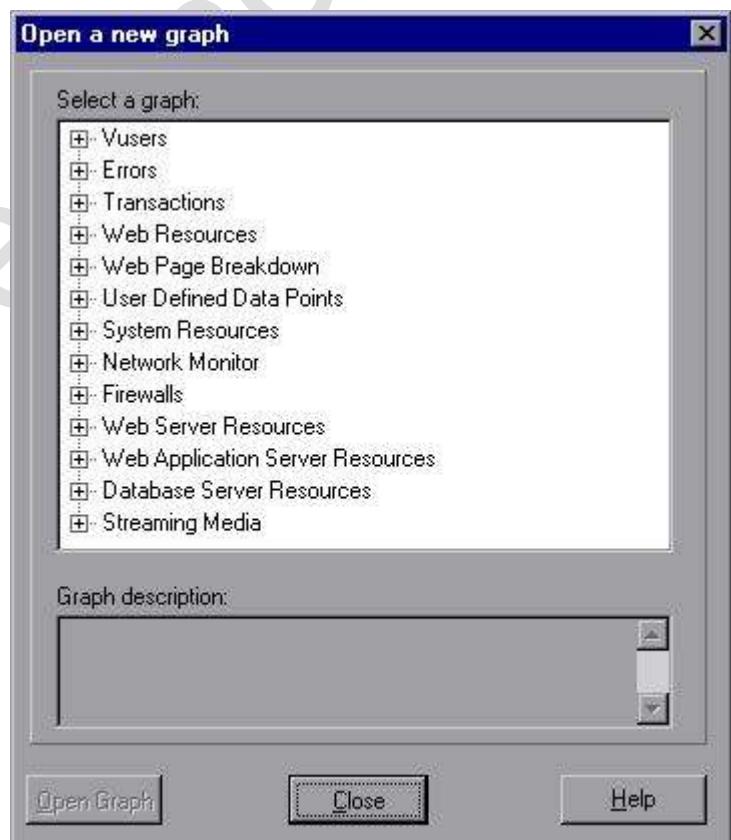
### Network Monitor

Network Monitor graphs provide information about the network delays.

# Opening Analysis graphs

By default, LoadRunner displays only the Summary Report in the graph tree view

- Select Graph > Add Graph
- or click <New Graph> in the graph tree view
- This opens a New Graph dialog box
- Choose the required graph



# Interpreting Analysis graphs

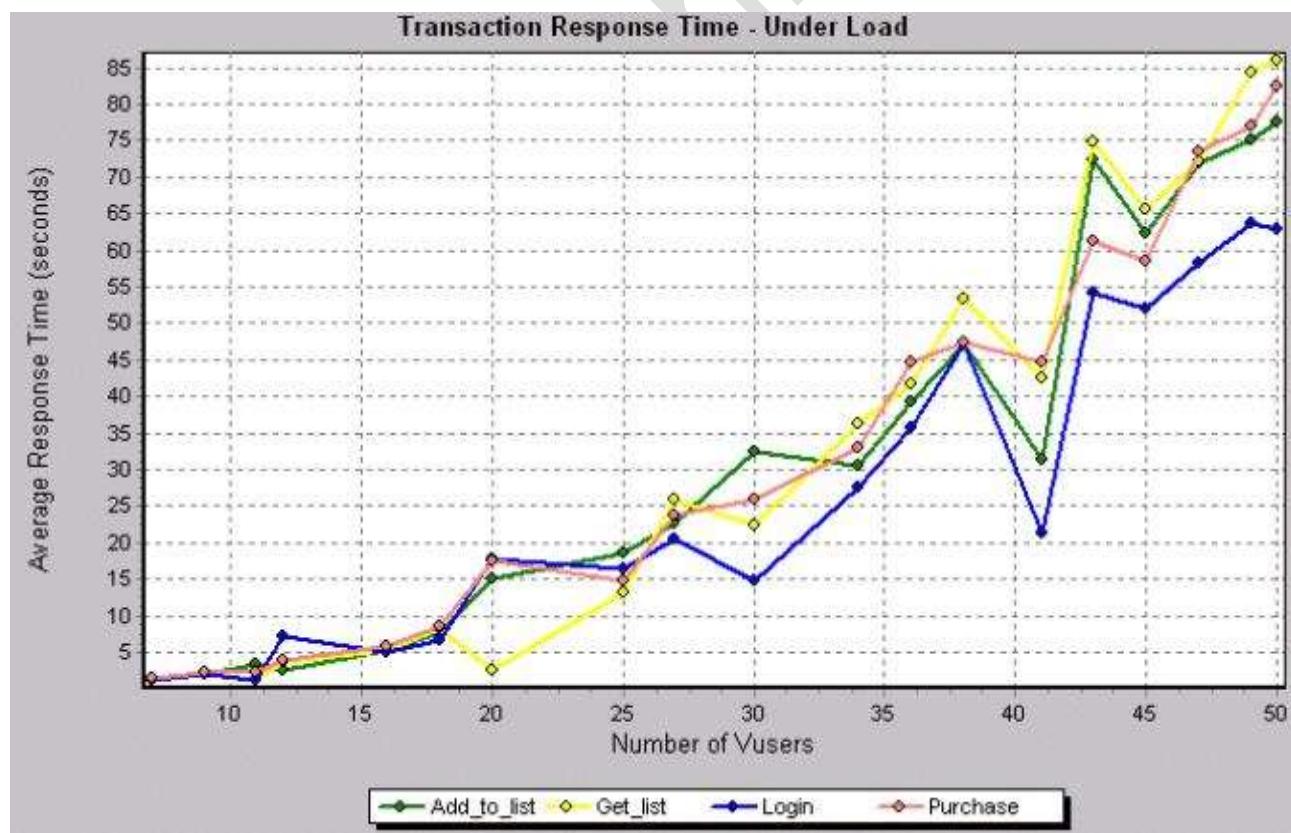
LoadRunner Analysis graphs present important information about the performance of your scenario.

Using these graphs,

- Can identify and pinpoint bottlenecks in applications
- Determine changes needed to improve its performance
- Estimate the maximum load the server can handle
- Find response time of transactions under varied load

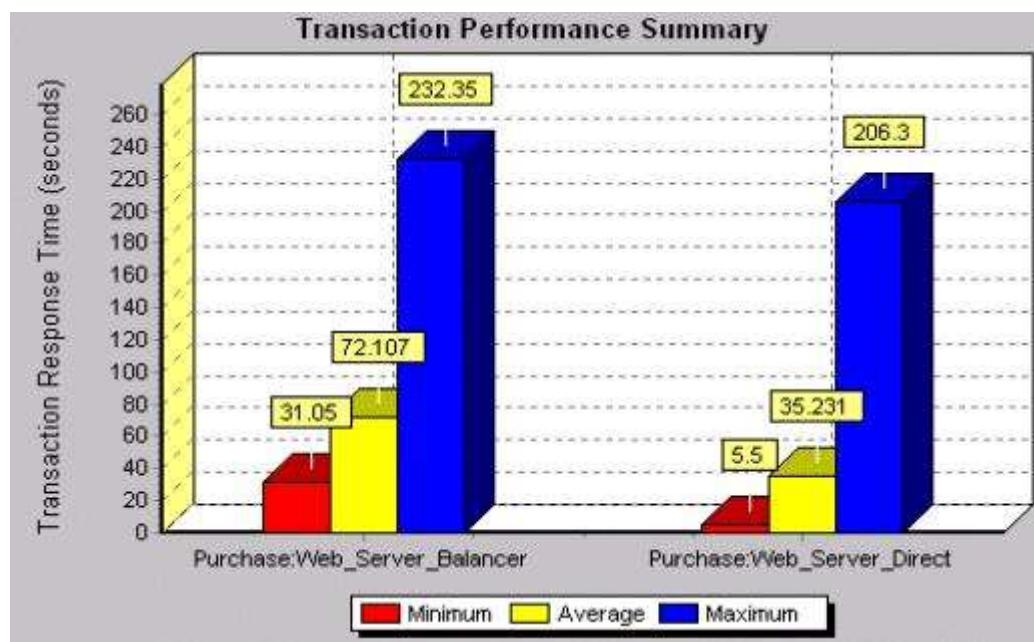
# Interpreting Analysis graphs

Case Study #1 Analyzing Performance under Load



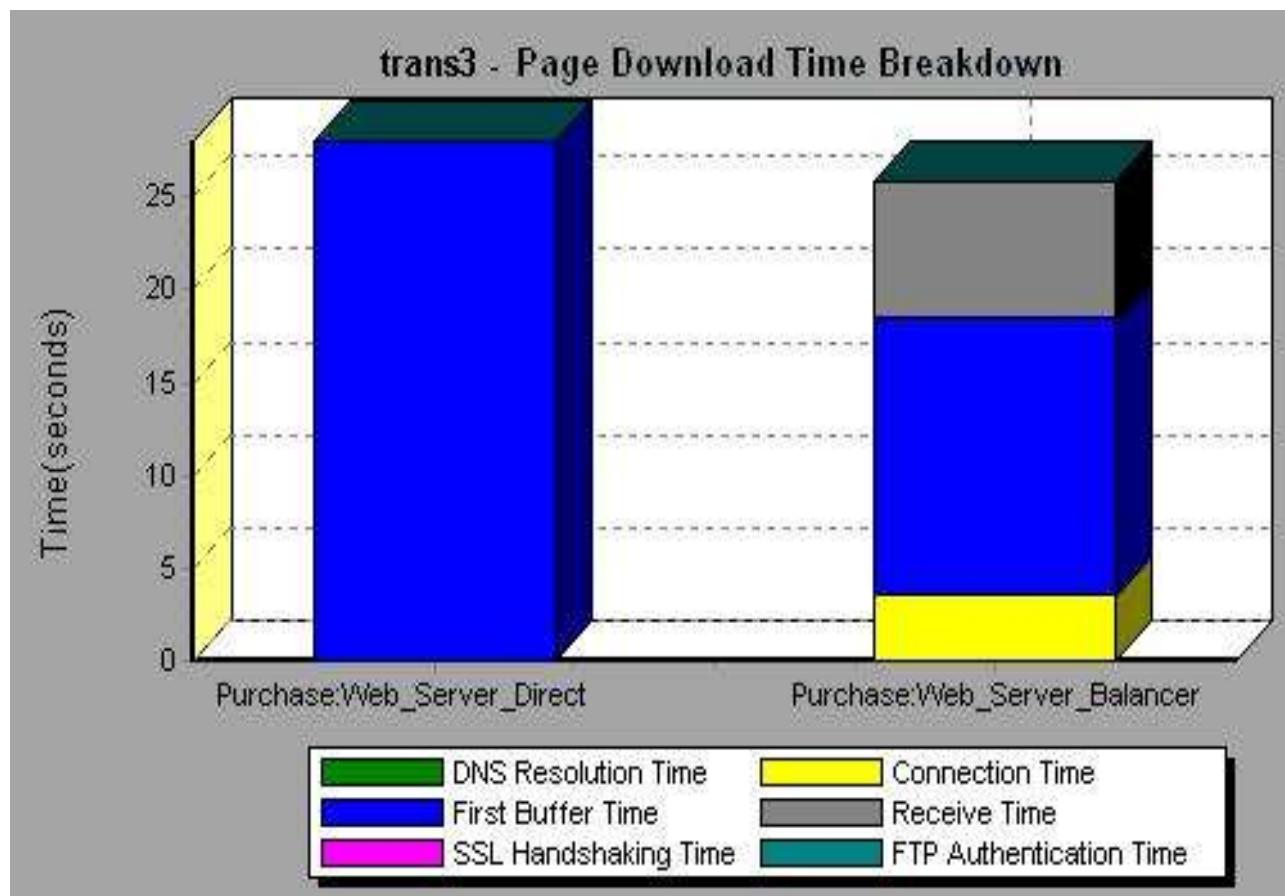
# Interpreting Analysis graphs

Case Study #2 Breaking down Trx Response time



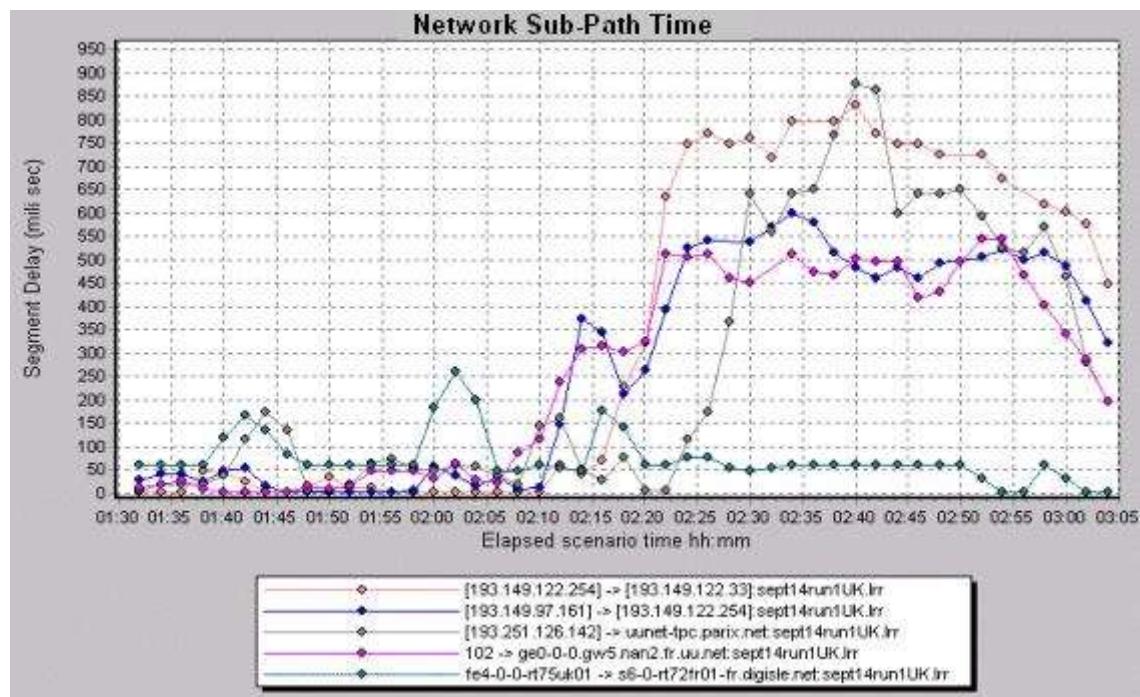
# Interpreting Analysis graphs

Case Study #3 Web page breakdown graphs



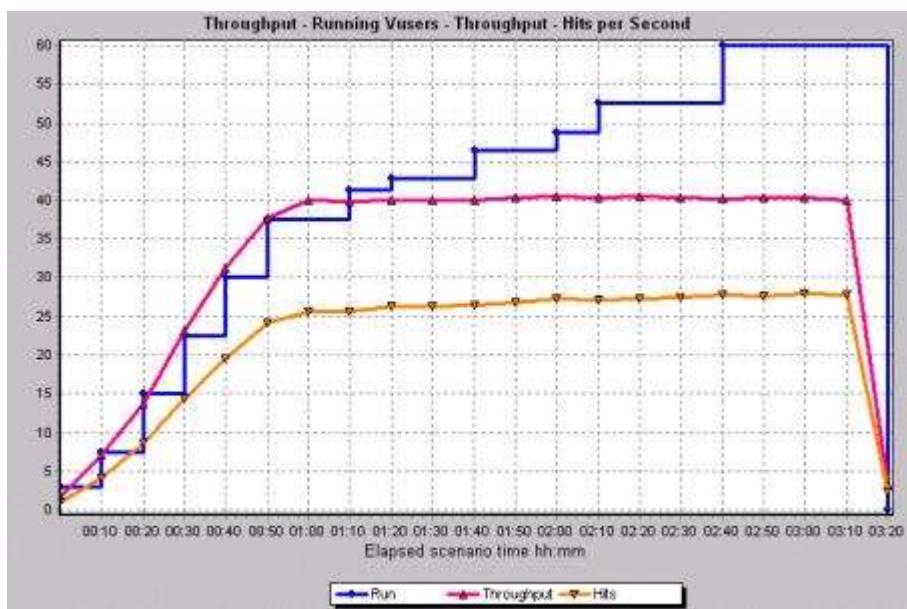
# Interpreting Analysis graphs

Case Study #4 Identifying Network Problems



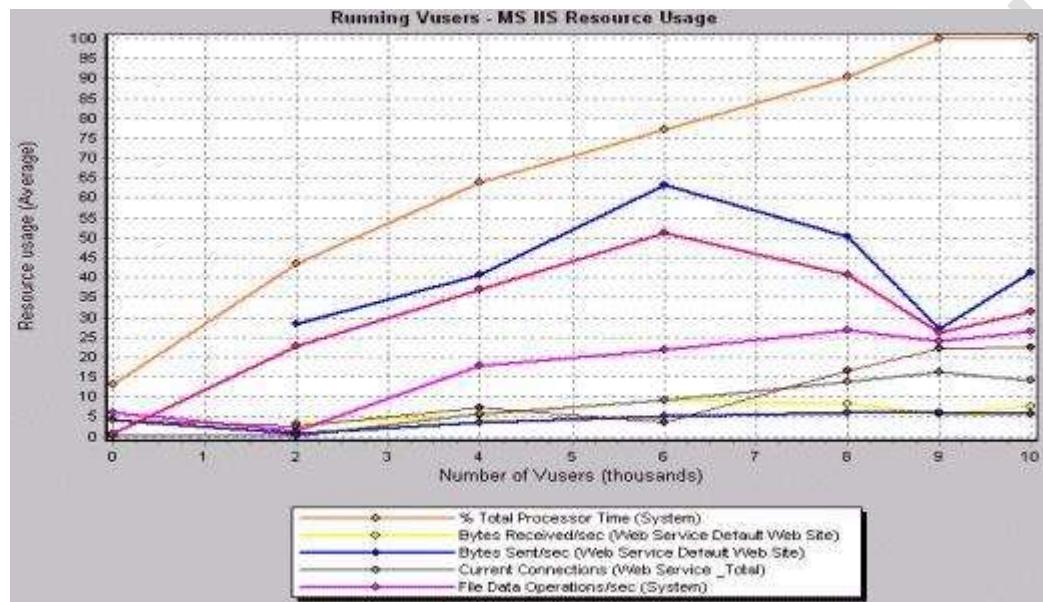
# Interpreting Analysis graphs

Case Study #5 Identifying Server Problems



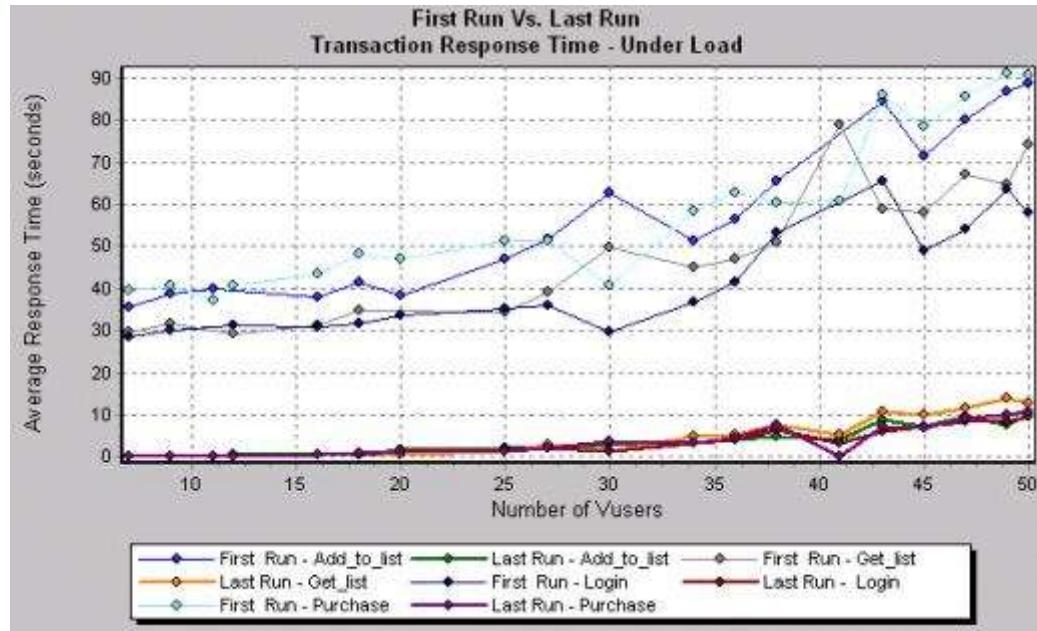
# Interpreting Analysis graphs

Case Study #5 Identifying Server Problems



# Interpreting Analysis graphs

Case Study #6 Comparing Scenario Results



## Summary

- Many concurrent users running the same application to see whether a system handles the load without compromising functionality or performance is known as Load Testing.
- Steps involved in Load Testing:
  - Planning Load Tests
  - Creating Vusers
  - Creating Scenarios
  - Executing Scenarios
  - Analysis of the system under Load

# References

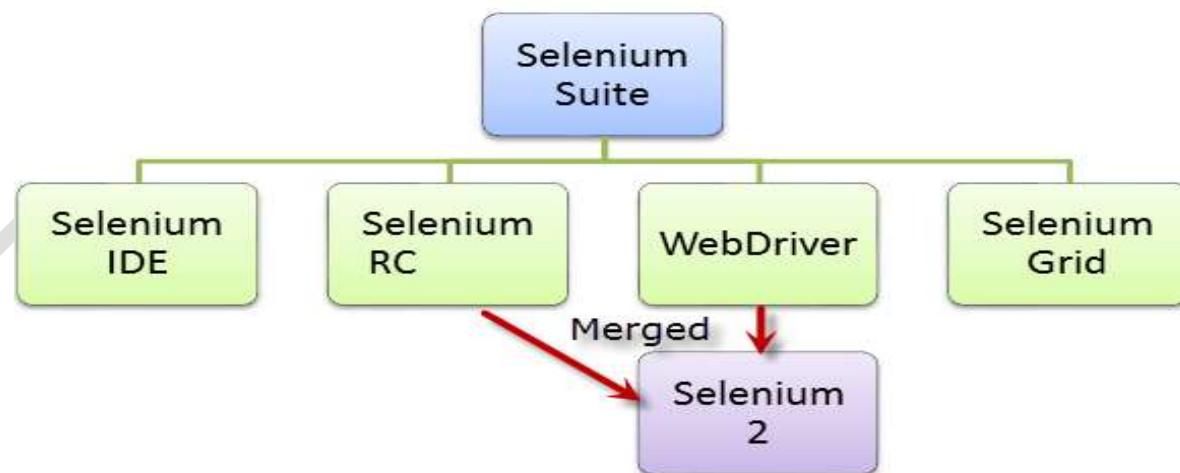
## Books :

- Effective Software Test Automation: Developing an Automated Software Testing Tool - Kanglin Li, Mengqi Wu, Sybex
- Automated Testing Handbook - Linda G. Hayes

# Selenium 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 re-named 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

TOPS Technologies

## 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.



Patrick Lightbody

## 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.



Simon Stewart

## Birth of Selenium2

- 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

## 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, through 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.

	Selenium IDE	Selenium RC	WebDriver
Browser Support	Mozilla Firefox	Mozilla Firefox Internet Explorer Google Chrome Safari Opera Konqueror Others	Internet Explorer versions 6 to 9, both 32 and 64-bit  Firefox 3.0, 3.5, 3.6, 4.0, 5.0, 6, 7 and above (current version is 16.0.1)  Google Chrome 12.0.712.0 and above (current version is 22.0.1229.94 m)  Opera 11.5 and above (current version is 12.02)  Android – 2.3 and above for phones and tablets (devices & emulators)  iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators)  HtmlUnit 2.9 and above (current version is 2.10)
Operating System	Windows Mac OS X Linux	Windows Mac OS X Linux Solaris	All operating systems where the browsers above can run.

## How to Choose the Right Selenium Tool for Your Need

Tool	Why Choose?
Selenium IDE	<ul style="list-style-type: none"> <li>• To learn about concepts on automated testing and Selenium, including:</li> <li>• Selenese commands such as type, open, clickAndWait, assert, verify, etc.</li> <li>• Locators such as id, name, xpath, css selector, etc.</li> <li>• Executing customized JavaScript code using runScript</li> <li>• Exporting test cases in various formats.</li> <li>• To create tests with little or no prior knowledge in programming.</li> <li>• To create simple test cases and test suites that you can export later to RC or WebDriver.</li> <li>• To test a web application against Firefox only.</li> </ul>
Selenium RC	<ul style="list-style-type: none"> <li>• To design a test using a more expressive language than Selenese</li> <li>• To run your test against different browsers (except HtmlUnit) on different operating systems.</li> <li>• To deploy your tests across multiple environments using Selenium Grid.</li> <li>• To test your application against a new browser that supports JavaScript.</li> <li>• To test web applications with complex AJAX-based scenarios.</li> </ul>
WebDriver	<ul style="list-style-type: none"> <li>• To use a certain programming language in designing your test case.</li> <li>• To test applications that are rich in AJAX-based functionalities.</li> <li>• To execute tests on the HtmlUnit browser.</li> <li>• To create customized test results.</li> </ul>
Selenium Grid	<ul style="list-style-type: none"> <li>• To run your Selenium RC scripts in multiple browsers and operating systems simultaneously.</li> <li>• To run a huge test suite, that need to complete in soonest time possible.</li> </ul>

## A Comparison between Selenium and QTP

Quick Test Professional(QTP) is a proprietary automated testing tool previously owned by the company Mercury Interactive before it was acquired by Hewlett-Packard in 2006. The Selenium Tool Suite has many advantages over QTP (as of version 11) as detailed below -

## 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 3rd party software
Can execute tests while the browser is minimized	Needs to have the application under test to be visible on the desktop
Can execute tests in parallel.	Can only execute in parallel but using Quality Center which is again a paid product.
<b>Can test both web and desktop applications</b>	Can only test web applications
<b>Comes with a built-in object repository</b>	Has no built-in object repository
<b>Automates faster than Selenium because it is a fully featured IDE.</b>	Automates at a slower rate because it does not have a native IDE and only third party IDE can be used for development
<b>Data-driven testing is easier to perform because it has built-in global and local data tables.</b>	Data-driven testing is more cumbersome since you have to rely on the programming language's capabilities for setting values for your test data
<b>Can access controls within the browser(such as the Favorites bar, Address bar, Back and Forward buttons, etc.)</b>	Cannot access elements outside of the web application under test
<b>Provides professional customer support</b>	No official user support is being offered.
<b>Has native capability to export test data into external formats</b>	Has no native capability to export runtime data onto external formats
<b>Parameterization Support is in built</b>	Parameterization can be done via programming but is difficult to implement.
<b>Test Reports are generated automatically</b>	No native support to generate test /bug reports.

Though clearly, QTP has more advanced capabilities, Selenium outweighs QTP in three main areas:

Cost(because Selenium is completely free)

Flexibility(because of a number of programming languages, browsers, and platforms it can support)

Parallel testing(something that QTP is capable of but only with use of Quality Center)

## **Summary**

The entire Selenium Tool Suite is comprised of four components:

Selenium IDE, a Firefox add-on that you can only use in creating relatively simple test cases and test suites.

Selenium Remote Control, also known as Selenium 1, which is the first Selenium tool that allowed users to use programming languages in creating complex tests.

WebDriver, the newer breakthrough that allows your test scripts to communicate directly to the browser, thereby controlling it from the OS level.

Selenium Grid is also a tool that is used with Selenium RC to execute parallel tests across different browsers and operating systems.

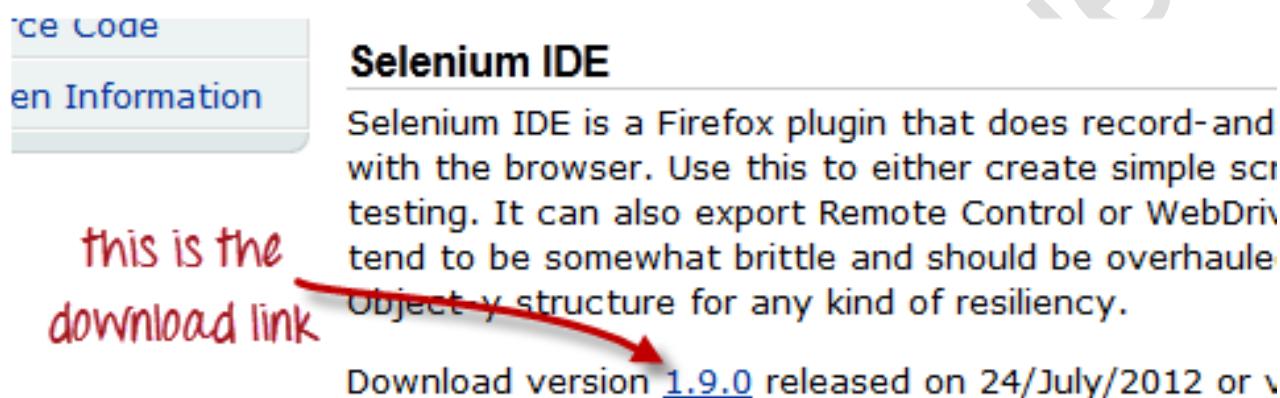
Selenium RC and WebDriver were merged to form Selenium 2.

Selenium is more advantageous than QTP in terms of costs and flexibility. It also allows you to run tests in parallel, unlike in QTP where you are only allowed to run tests sequentially.

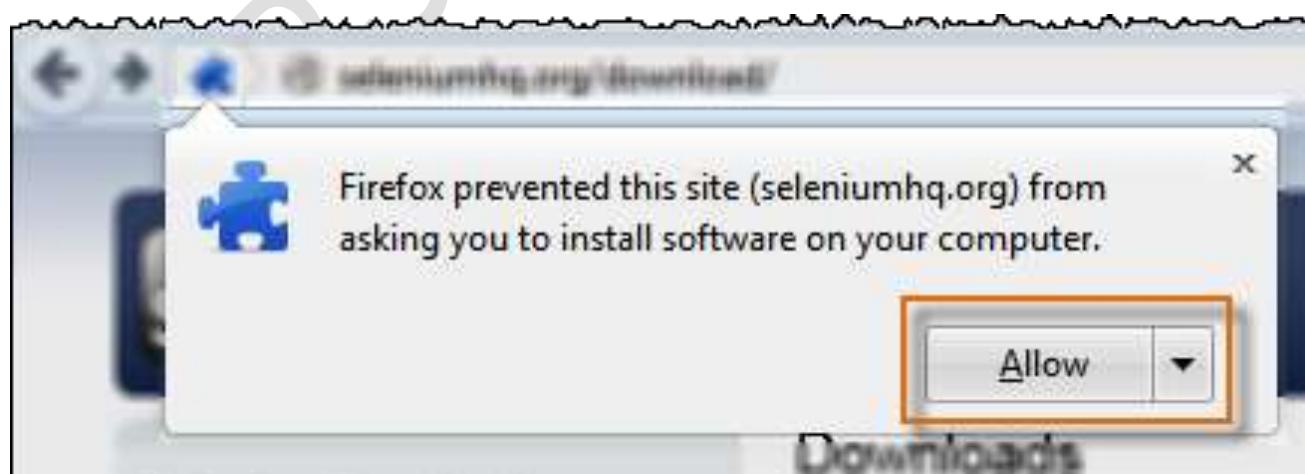
# Installing Selenium IDE & FireBug

## Steps

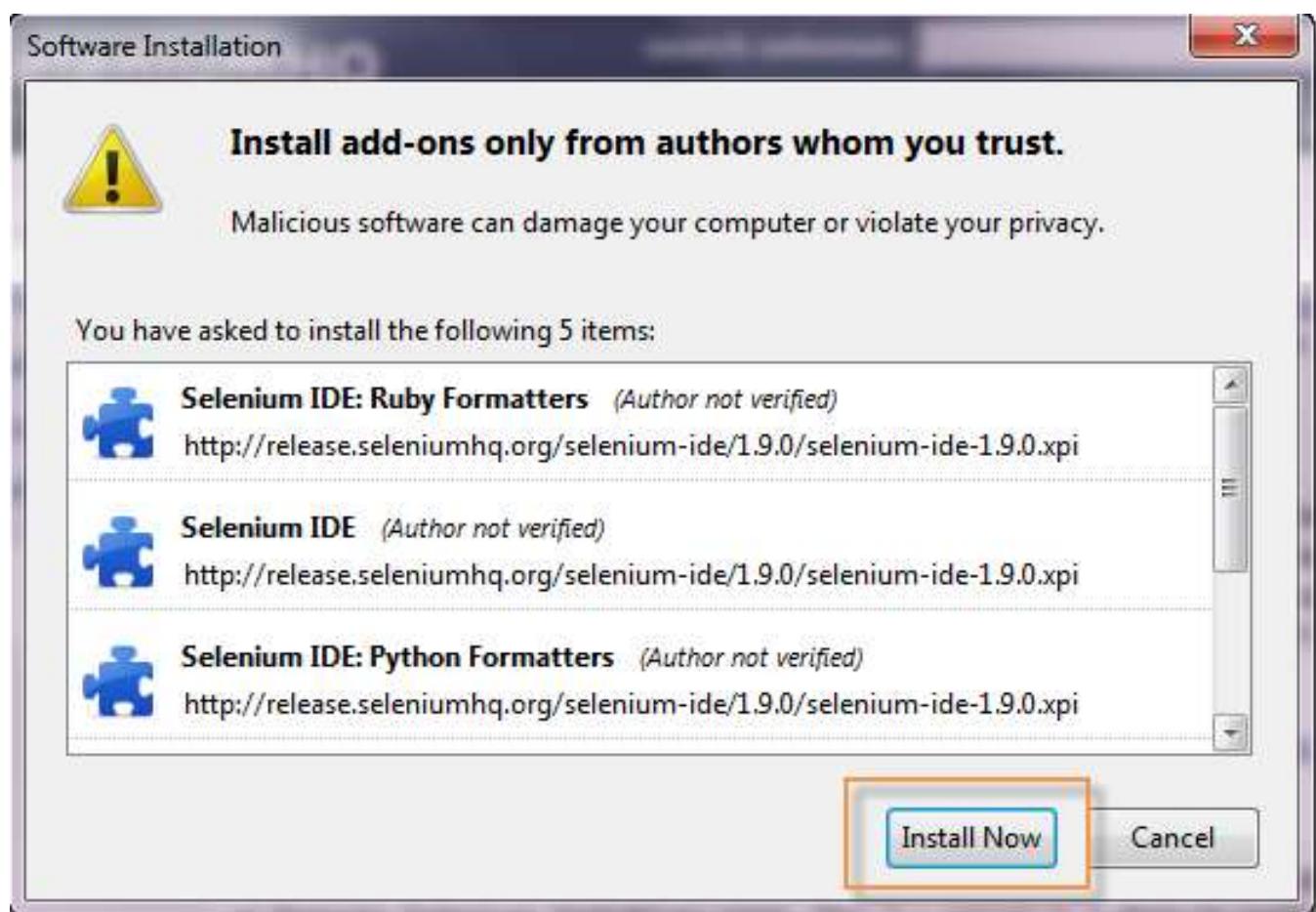
Launch Firefox and navigate to <http://seleniumhq.org/download/>. Under the Selenium IDE section, click on the link that shows the current version number.



For security, a Firefox notification will pop up. Click on “Allow.”

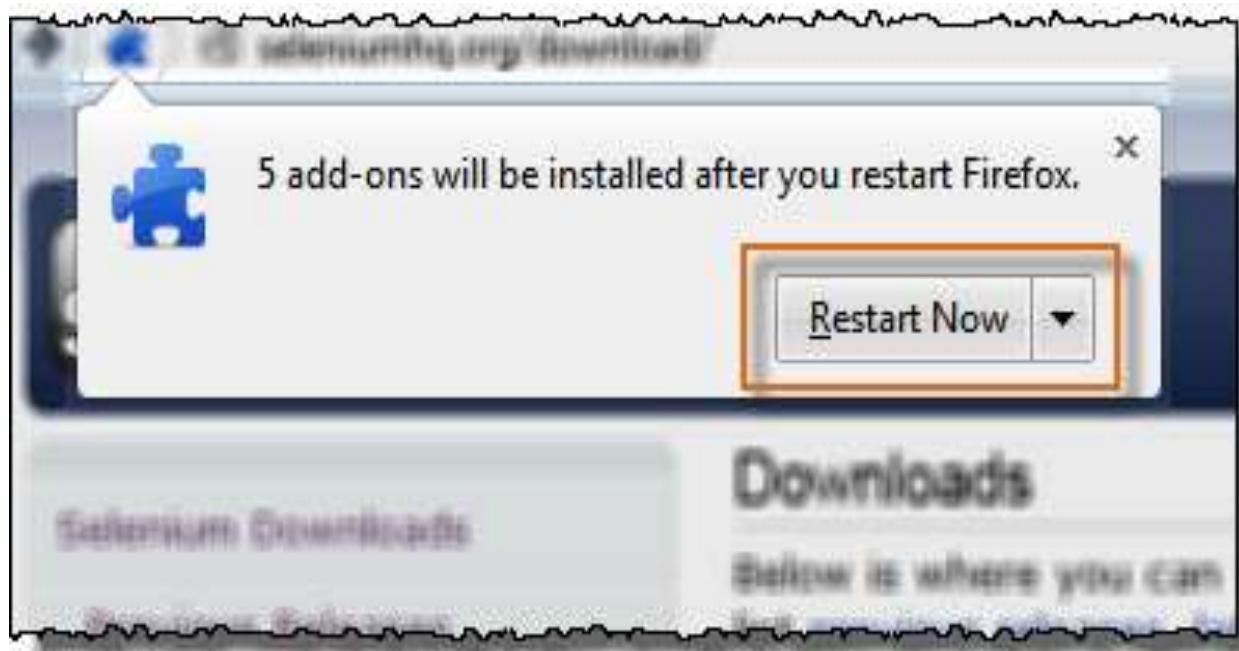


Wait until Firefox completes the download and then click “Install Now.”



TOPS

Wait until installation is completed. In the pop-up window, click “Restart Now.”

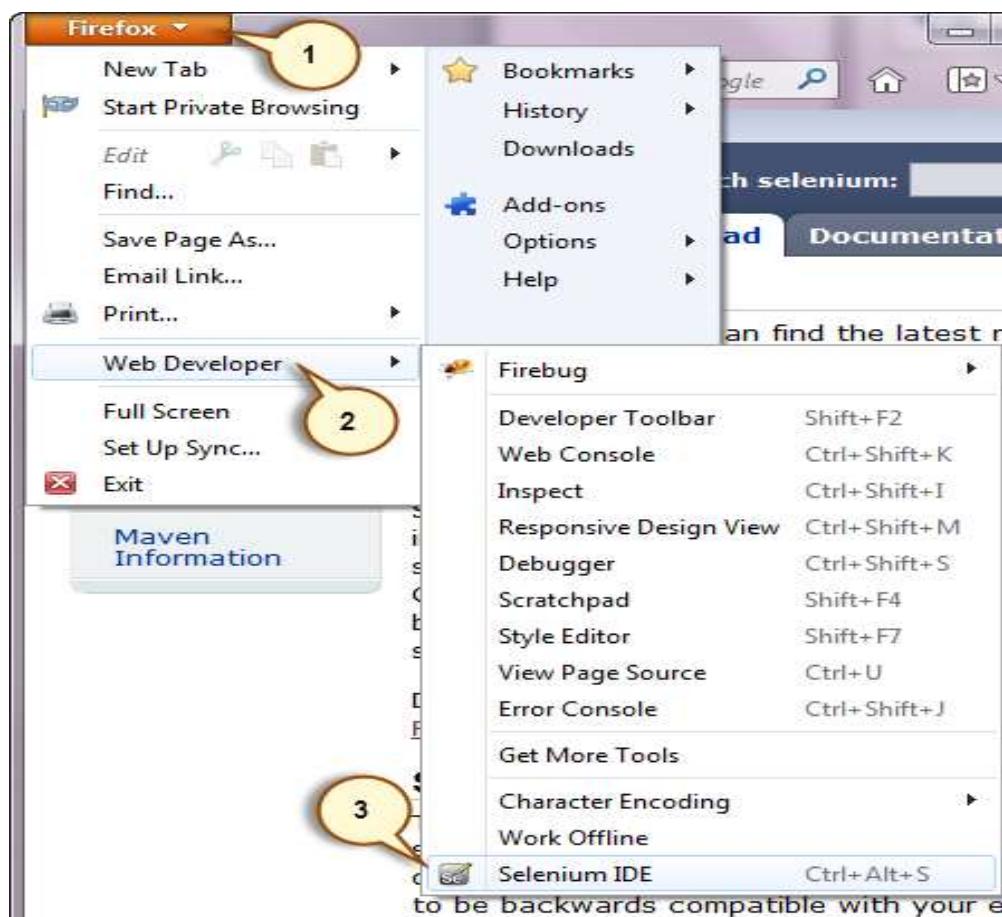


TOPS Technologies

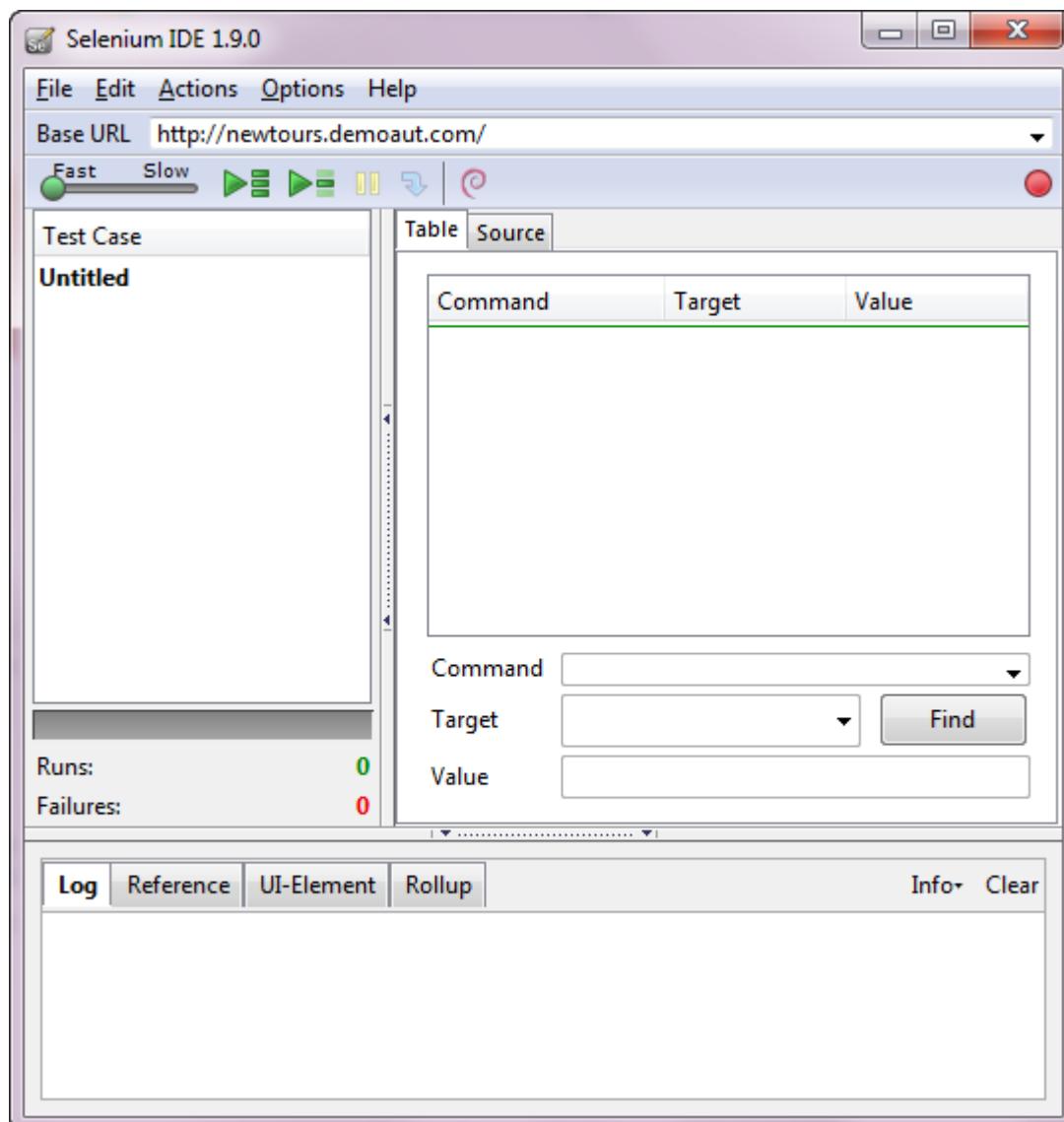
After Firefox has restarted, launch Selenium IDE using either of two ways:

By pressing **Ctrl+Alt+S**

By clicking on the Firefox menu button > Web Developer> Selenium IDE



Selenium IDE should launch as shown below



# Installation of Firebug

Firebug is a Firefox add-on that we will use to inspect the HTML elements of the web application under test. It will provide us the name of the element that our Selenese command would act upon.

## Step 1

Use Firefox to navigate to Firebug's download page (<https://getfirebug.com/downloads/>) and click on the download link.



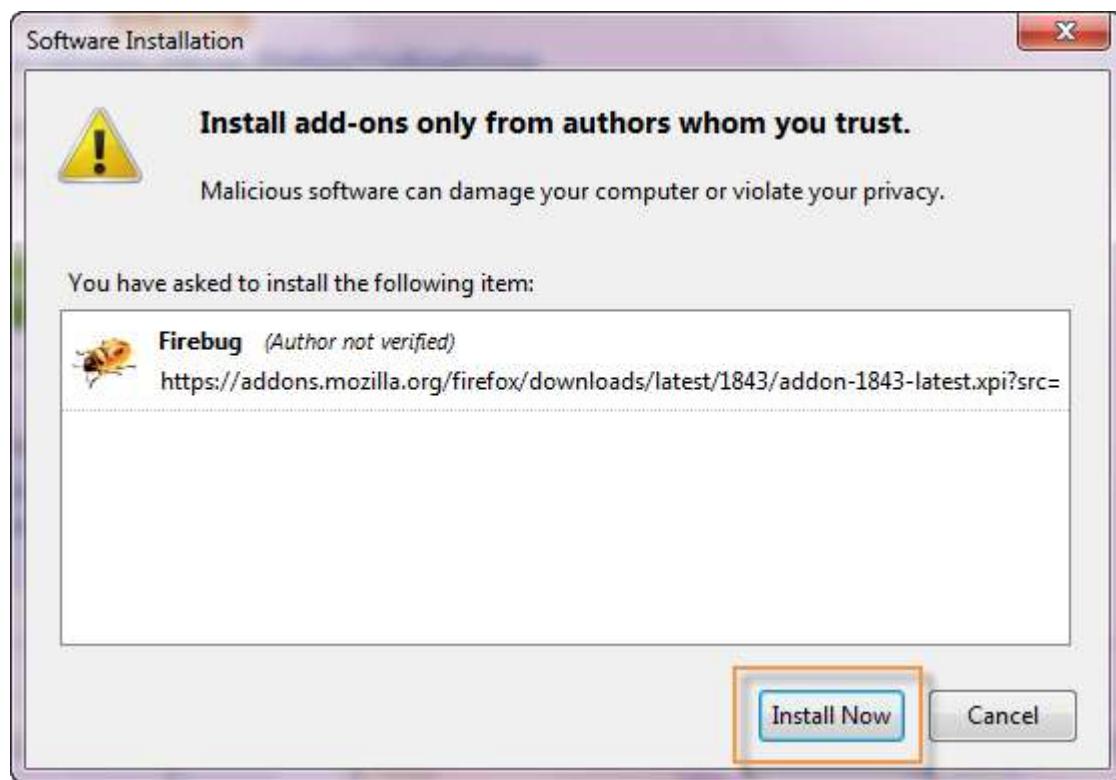
## Step 2

Firefox will take you to its Firebug download section. Click the “Add to Firefox” button.



## Step 3

Wait for Firefox to complete downloading this add-on. On the dialog box that comes after, click “Install Now.”



## Step 4

Wait for installation to complete. A notification will pop-up saying, “Firebug has been installed successfully.” You can immediately close this pop-up.



Note: In case you do not see above pop-up , no worries! This pop-up appears for a few seconds and disappears.

## Step 5

Launch Firebug by doing either of these two methods:

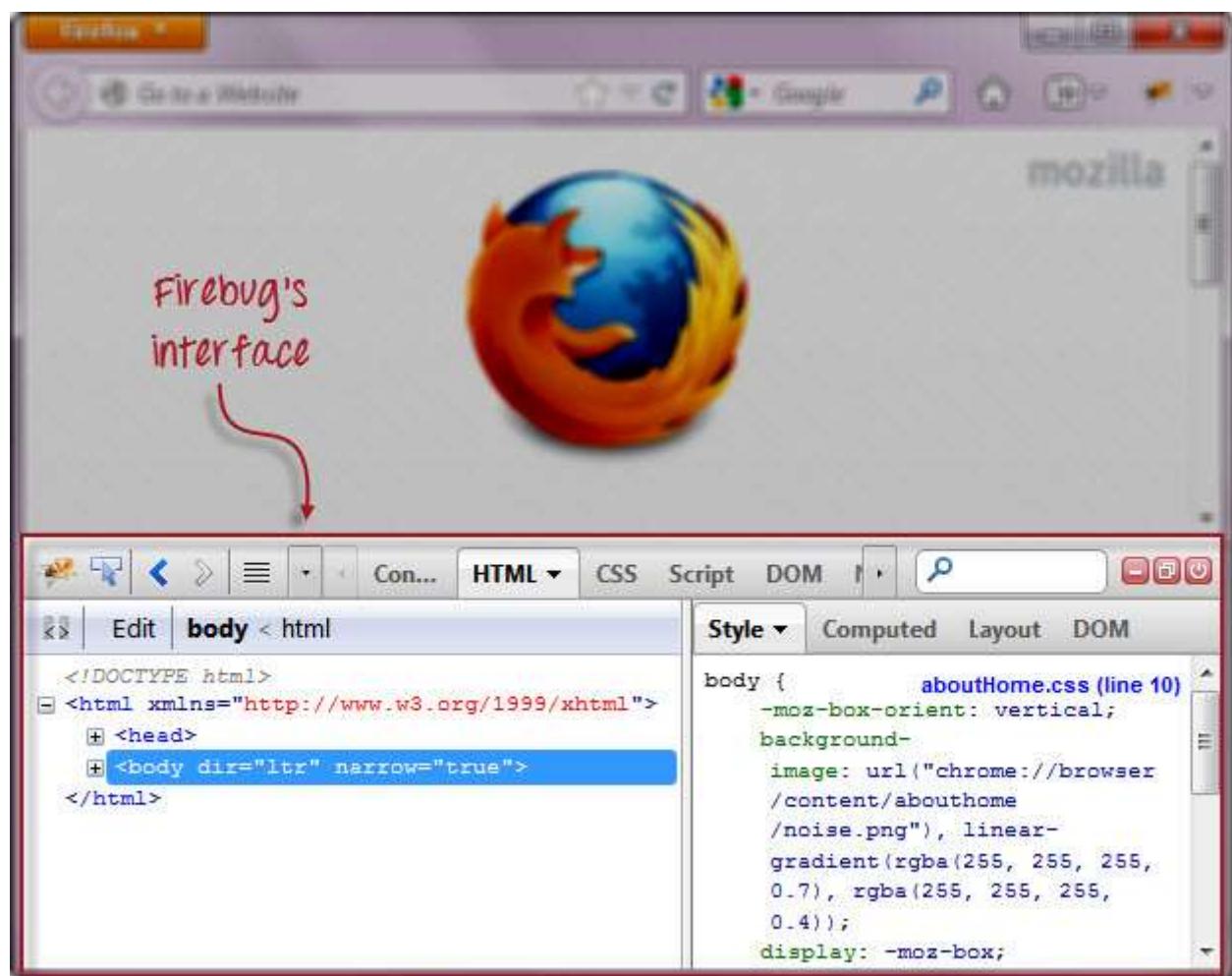
Press F12

Click on the Firebug button on the upper right corner of the Firefox window.



## Step 6

Firebug should launch at the bottom of Firefox as shown below



## Plugins

Selenium IDE can support additional Firefox add-ons or plugins created by other users. You can visit [here](https://addons.mozilla.org/en-US/firefox/search/?q=selenium&appver=16.0&platform=windows) for a list of Selenium add-ons available to date. Install them just as you do with other Firefox add-ons.

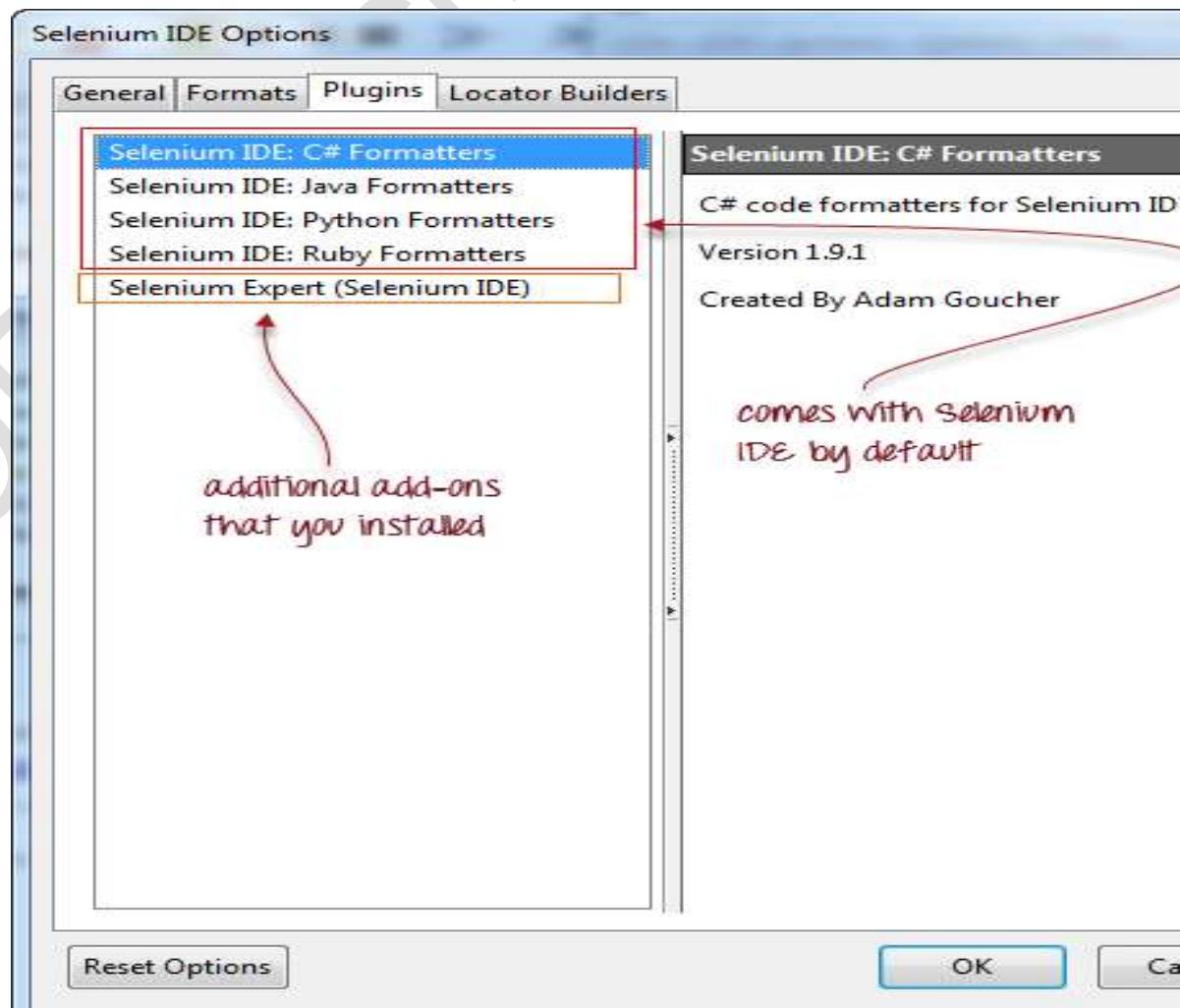
(<https://addons.mozilla.org/en-US/firefox/search/?q=selenium&appver=16.0&platform=windows>)

By default, Selenium IDE comes bundled with 4 plugins:

1. Selenium IDE: C# Formatters
2. Selenium IDE: Java Formatters
3. Selenium IDE: Python Formatters
4. Selenium IDE: Ruby Formatters

These four plugins are required by Selenium IDE to convert Selenese into different formats.

The Plugins tab shows a list of all your installed add-ons, together with the version number and name of the creator of each.

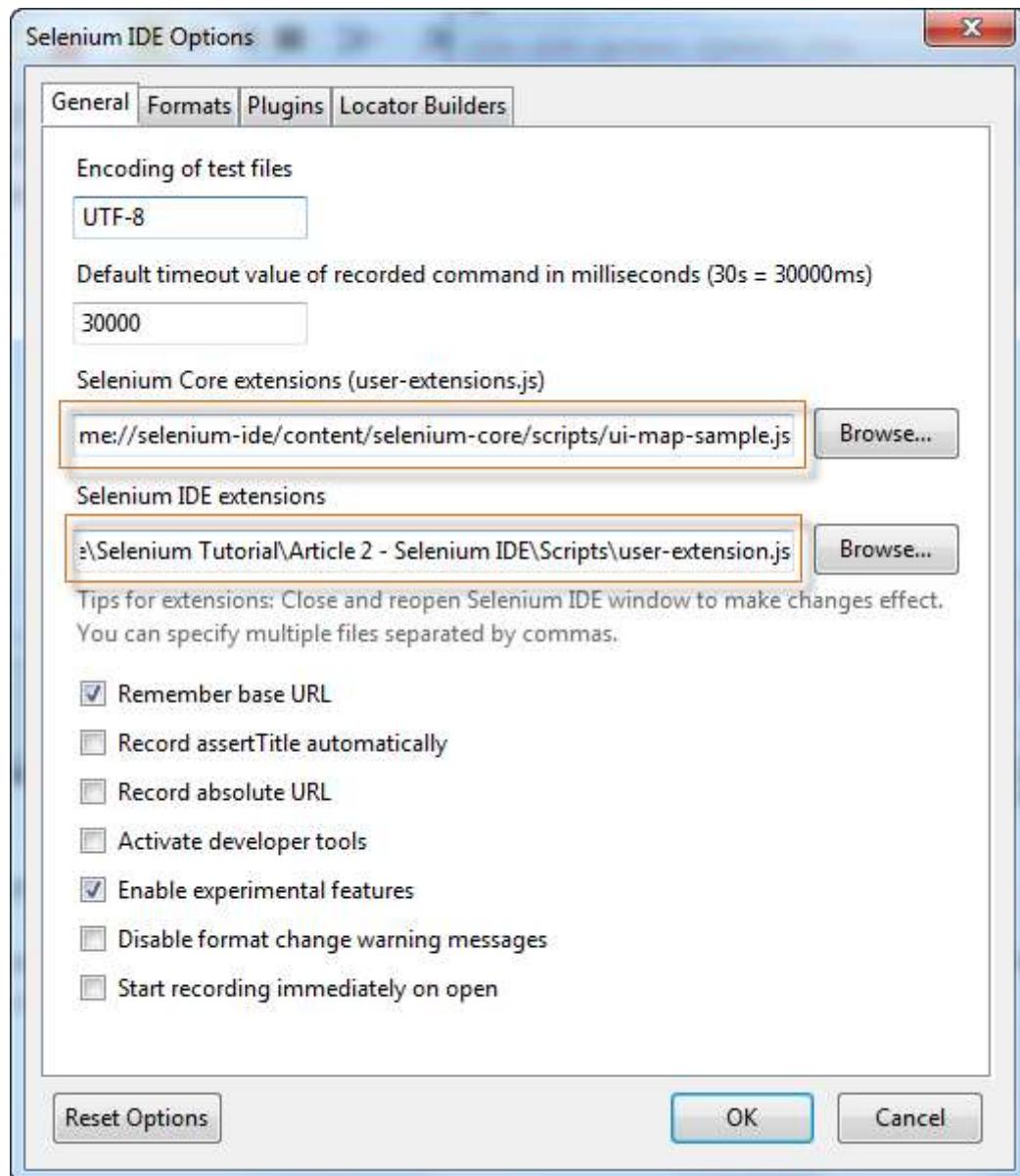


## User Extensions

Selenium IDE can support user extensions to provide advanced capabilities. User extensions are in the form of JavaScript files. You install them by specifying their absolute path in either of these two fields in the Options dialog box.

### Selenium Core extensions (user-extensions.js)

#### Selenium IDE extensions



## Selenium IDE

**Selenium IDE (Integrated Development Environment) is the simplest tool in the Selenium Suite. It is a Firefox add-on that creates tests very quickly through its record-and-playback functionality. This feature is similar to that of QTP. It is effortless to install and easy to learn.**

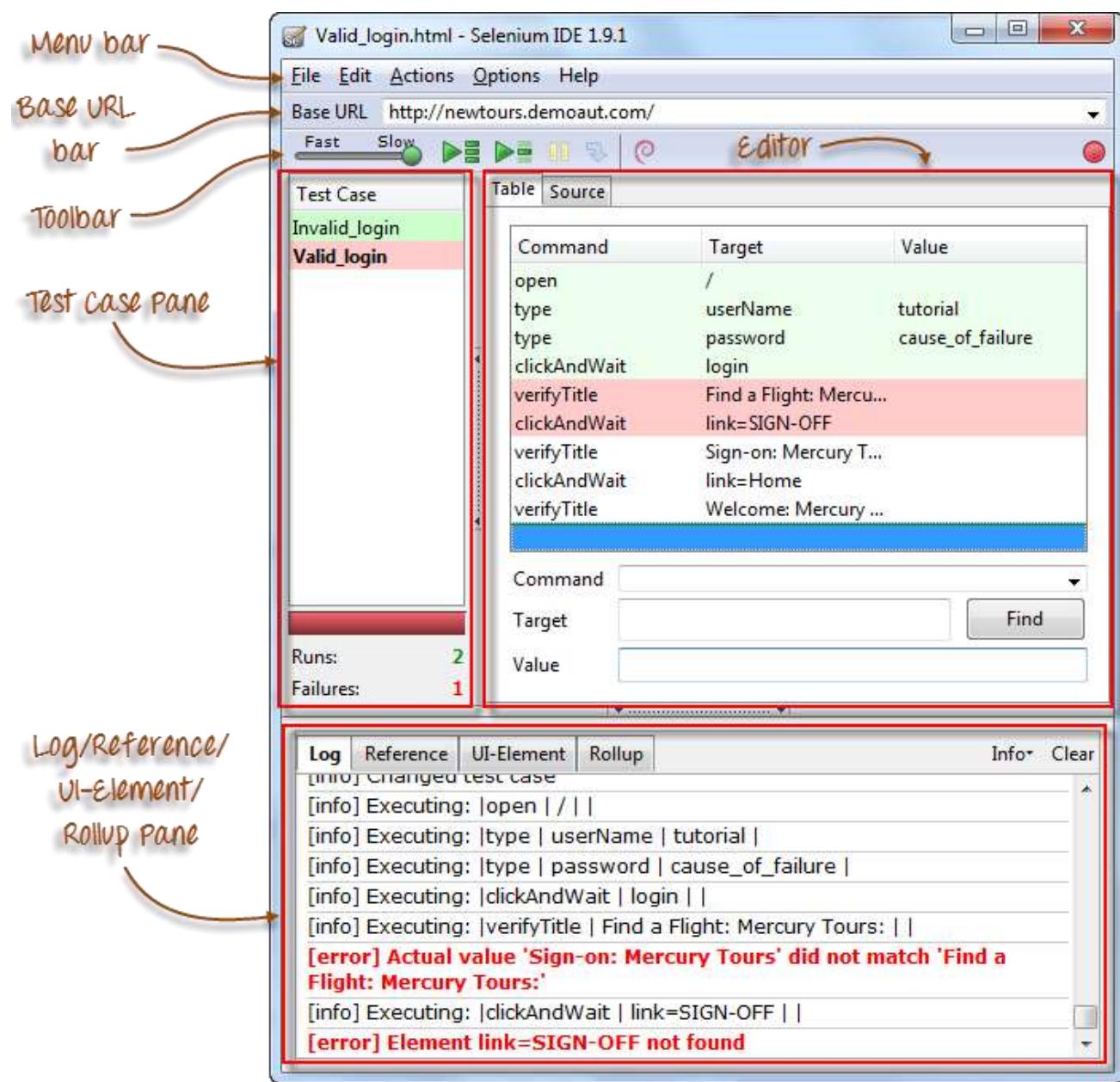
**Because of its simplicity, Selenium IDE should only be used as a prototyping tool – not an overall solution for developing and maintaining complex test suites.**

**Though you will be able to use Selenium IDE without prior knowledge in programming, you should at least be familiar with HTML, JavaScript, and the DOM (Document Object Model) to utilize this tool to its full potential. Knowledge of JavaScript will be required when we get to the section about the Selenese command “runScript”.**

**Selenium IDE supports auto complete mode when creating tests. This feature serves two purposes:**

**It helps the tester to enter commands more quickly.**

**It restricts the user from entering invalid commands.**



## Menu Bar

It is located at the topmost portion of the IDE. The most commonly used menus are the File, Edit, and Options menus.

### File menu

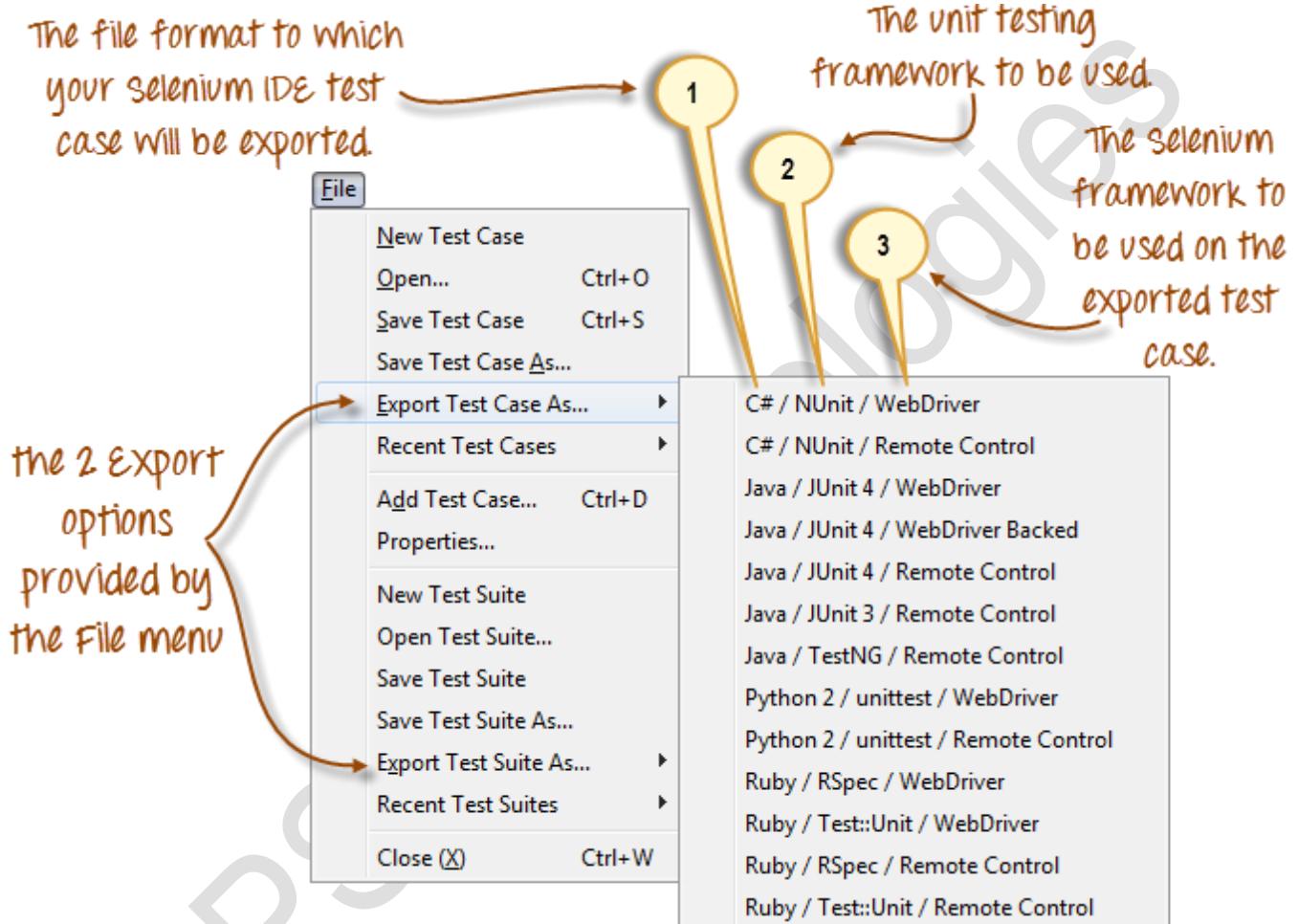
It contains options to create, open, save, and close tests.

Tests are saved in HTML format.

The most useful option is “Export” because it allows you to turn your Selenium IDE test cases into file formats that can run on Selenium Remote Control and WebDriver

“Export Test Case As...” will export only the currently opened test case.

“Export Test Suite As...” will export all the test cases in the currently opened test suite.



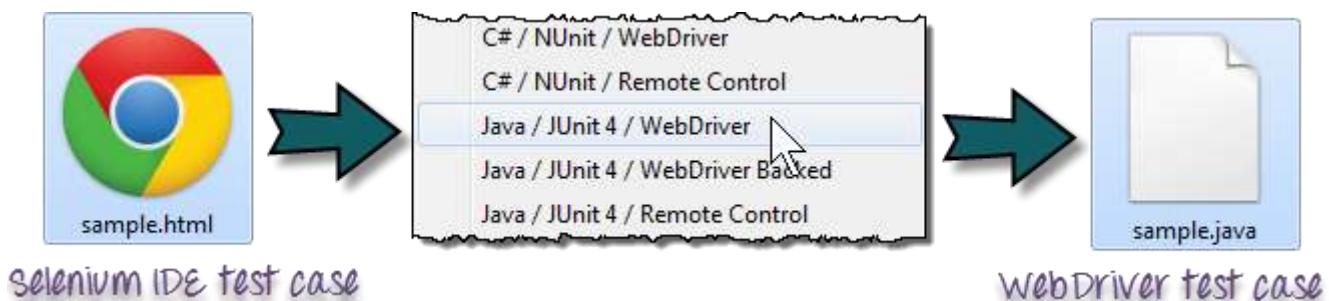
As of Selenium IDE v1.9.1, test cases can be exported only to the following formats:

.cs (C# source code)

.java (Java source code)

.py (Python source code)

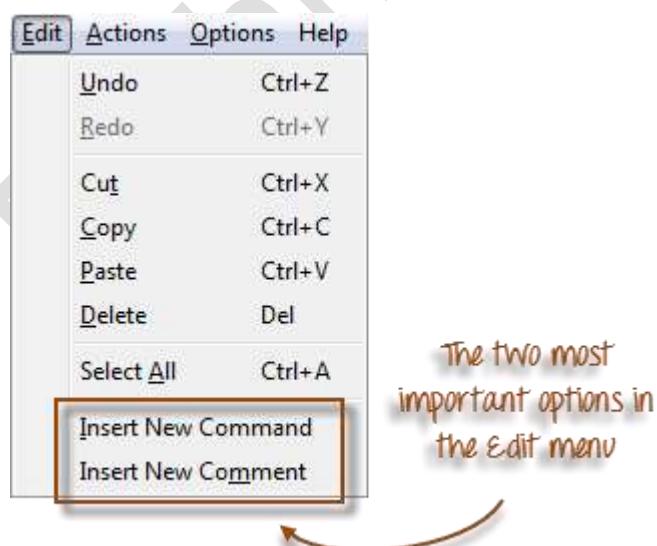
.rb (Ruby source code)



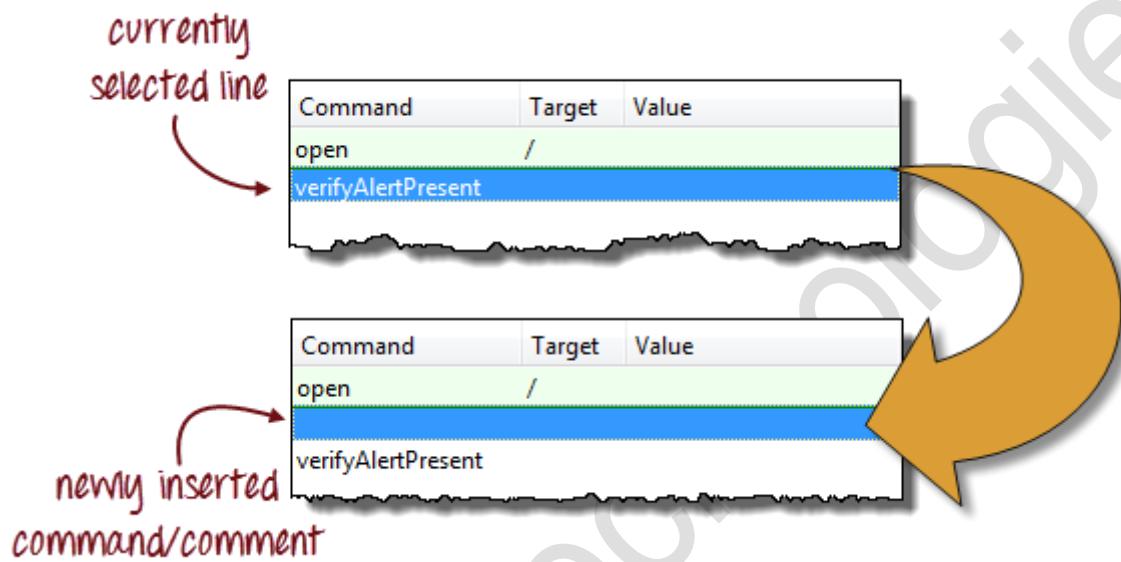
## Edit Menu

It contains usual options like **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Delete**, and **Select All**.

The two most important options are the “Insert New Command” and “Insert New Comment”.

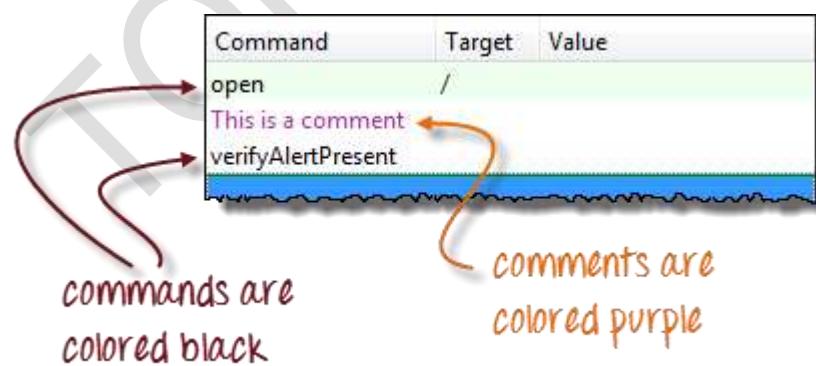


The newly inserted command or comment will be placed on top of the currently selected line.



Commands are colored black.

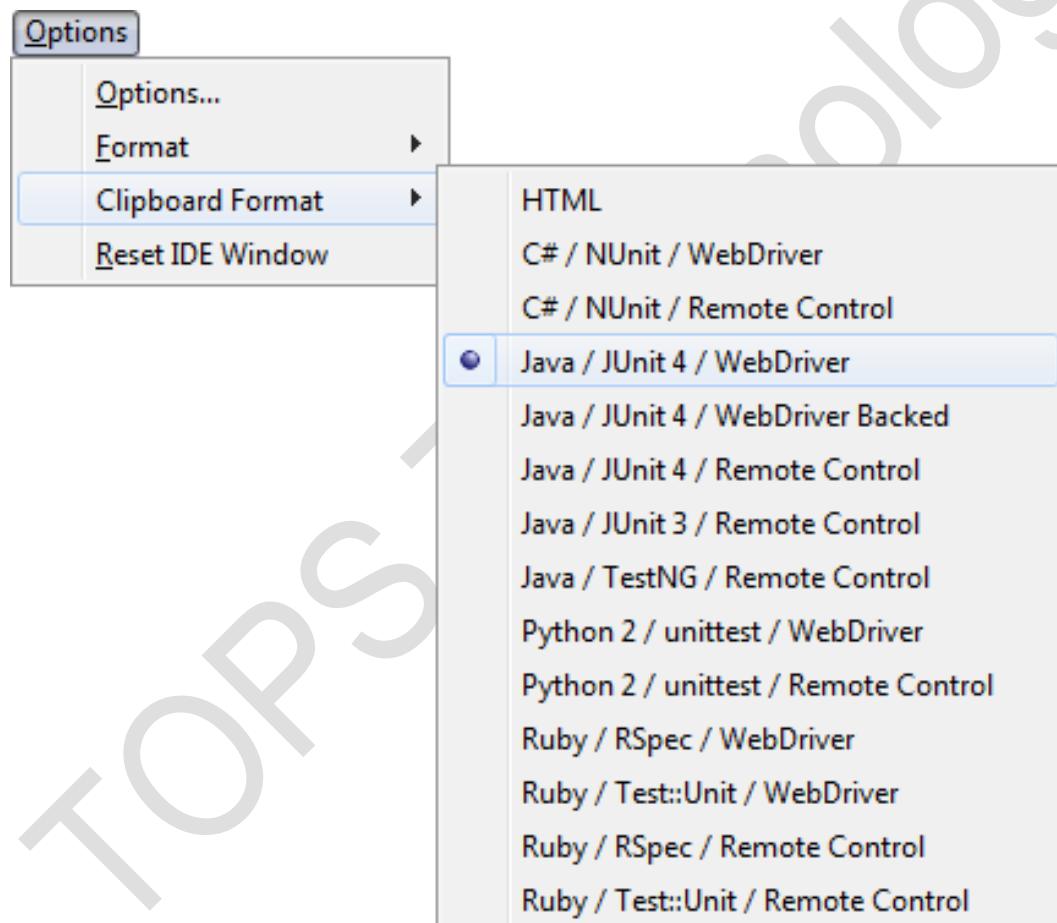
Comments are colored purple.



## Options menu

It provides the interface for configuring various settings of Selenium IDE.

We shall concentrate on the Options and Clipboard Format options.



For example, when you choose Java/JUnit 4/WebDriver as your clipboard format, every Selenese command you copy from Selenium IDE's editor will be pasted as Java code. See the illustration below.

Command	Target	Value
type	name=username	tutorial

When you copy this single line of Selenese command, it will be pasted as two lines of Java code on your favorite Java IDE

```
138
139     driver.findElement(By.name("username")).clear();
140     driver.findElement(By.name("username")).sendKeys("tutorial");
141
```

# Selenium IDE Options dialog box

You can launch the Selenium IDE Options dialog box by clicking Options > Options... on the menu bar. Though there are many settings available, we will concentrate on the few important ones.

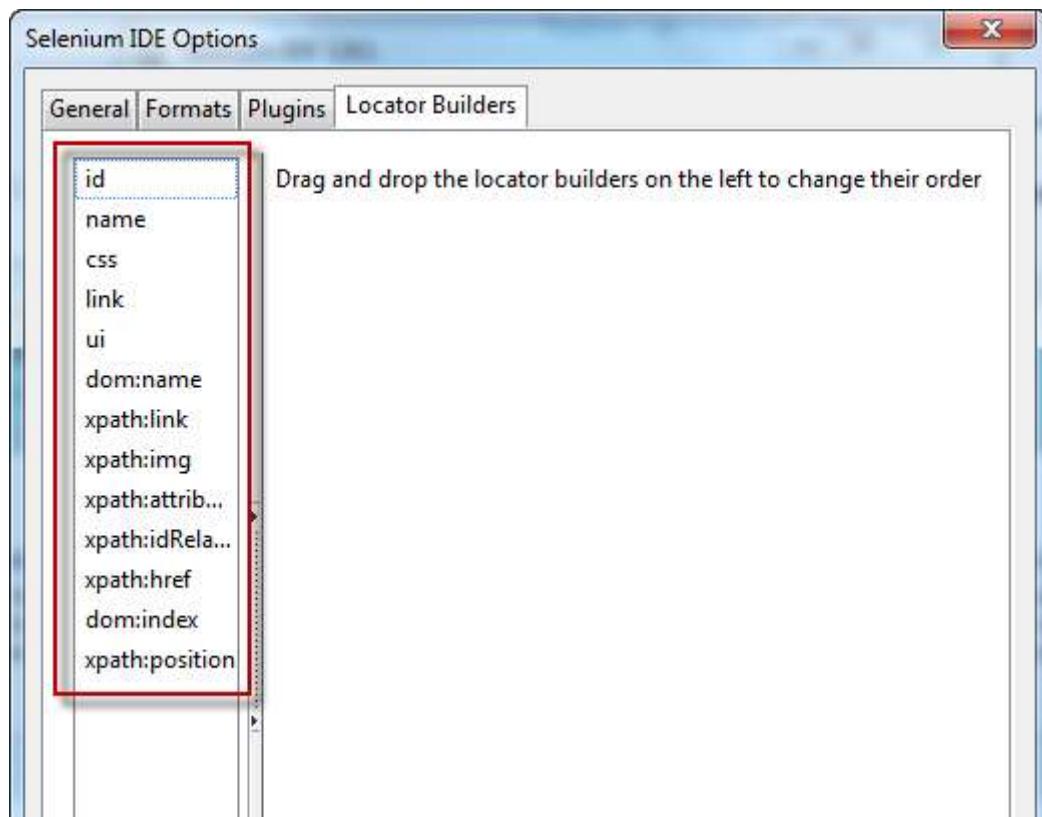
**Default Timeout Value.** This refers to the time that Selenium has to wait for a certain element to appear or become accessible before it generates an error. Default timeout value is 30000ms.

**Selenium IDE extensions.** This is where you specify the extensions you want to use to extend Selenium IDE's capabilities. You can visit <http://addons.mozilla.org/en-US/firefox/> and use "Selenium" as keyword to search for specific extensions.

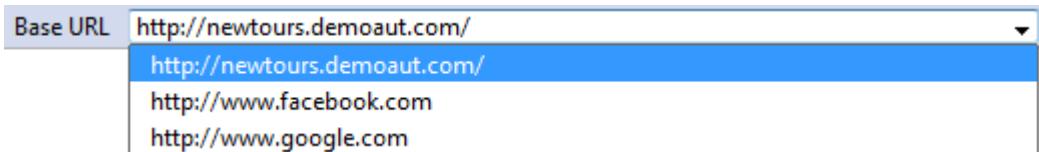
**Remember base URL.** Keep this checked if you want Selenium IDE to remember the Base URL every time you launch it. If you uncheck this, Selenium IDE will always launch with a blank value for the Base URL.

**Autostart record.** If you check this, Selenium IDE will immediately record your browser actions upon startup.

**Locator builders.** This is where you specify the order by which locators are generated while recording. Locators are ways to tell Selenium IDE which UI element should a Selenese command act upon. In the setup below, when you click on an element with an ID attribute, that element's ID will be used as the locator since "id" is the first one in the list. If that element does not have an ID attribute, Selenium will next look for the "name" attribute since it is second in the list. The list goes on and on until an appropriate one is found.



## Base URL Bar



It has a dropdown menu that remembers all previous values for easy access.

The Selenese command “open” will take you to the URL that you specified in the Base URL.

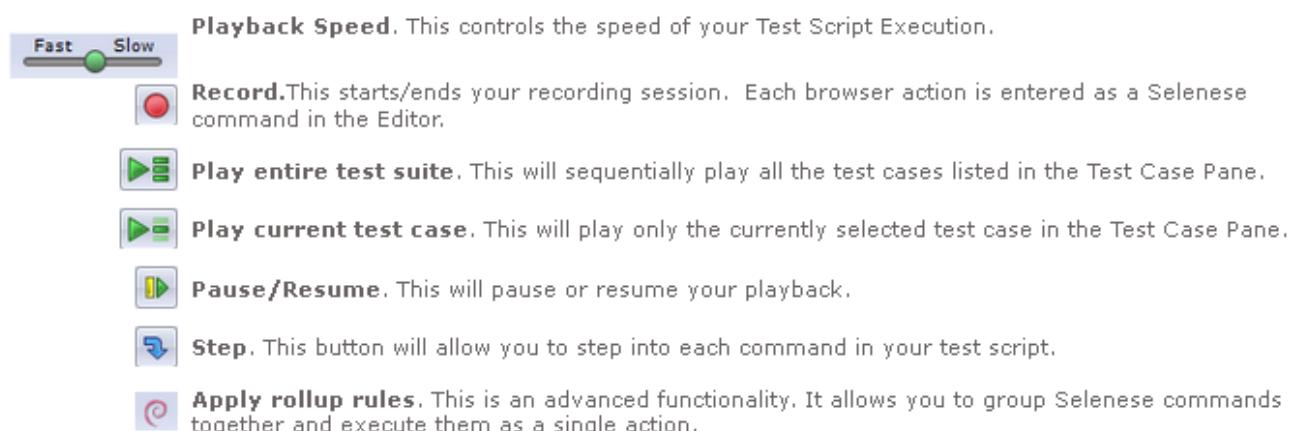
In this tutorial series, we will be using <http://newtours.demoaut.com> as our Base URL. It is the site for Mercury Tours, a web application maintained by HP for web testing purposes. We shall be using this application because it contains a complete set of elements that we need for the succeeding topics.

The Base URL is very useful in accessing relative URLs. Suppose that your Base URL is set to <http://newtours.demoaut.com>. When you execute the command “open” with the target value “signup”, Selenium IDE will direct the browser to <http://newtours.demoaut.com/signup>. See the illustration below.



TOPS Techies

## Toolbar



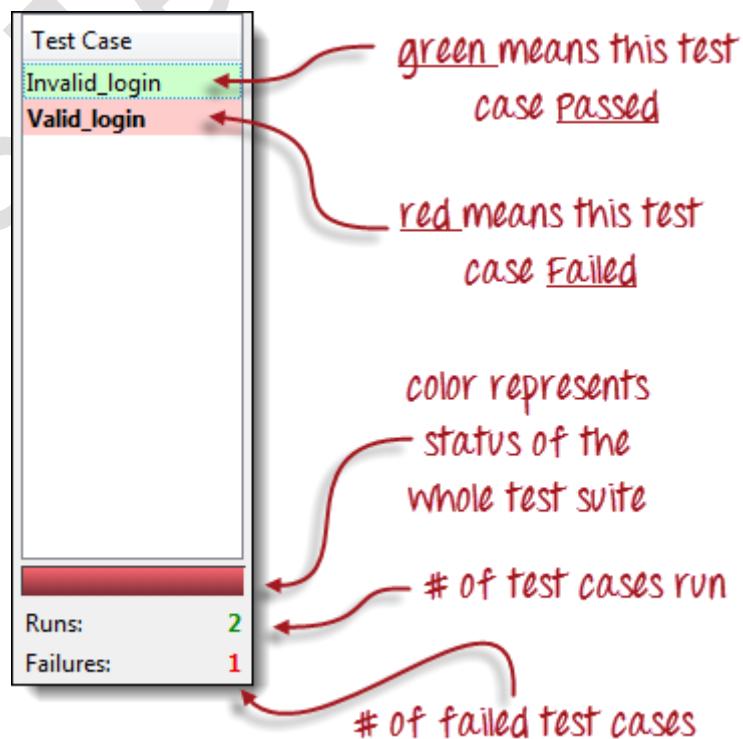
In Selenium IDE, you can open more than one test case at a time.

The test case pane shows you the list of currently opened test cases.

When you open a test suite, the test case pane will automatically list all the test cases contained in it.

The test case written in bold font is the currently selected test case

After playback, each test case is color-coded to represent if it passed or failed.



Green color means "Passed."

Red color means "Failed."

At the bottom portion is a summary of the number of test cases that were run and failed.

TOPS Technologies

# Editor

You can think of the editor as the place where all the action happens. It is available in two views: Table and Source.

Command	Target	Value
open	/	
type	userName	tutorial
type	password	tutorial
clickAndWait	login	
verifyTitle	Find a Flight: Mercu...	
clickAndWait	link=SIGN-OFF	
verifyTitle	Sign-on: Mercury T...	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury ...	

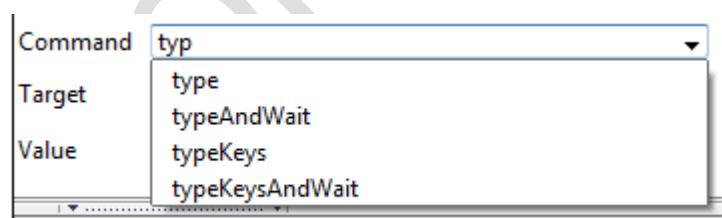
Command:   
 Target:    
 Value:

Table View

Most of the time, you will work on Selenium IDE using the Table View.

This is where you create and modify Selenese commands.

After playback, each step is color-coded.

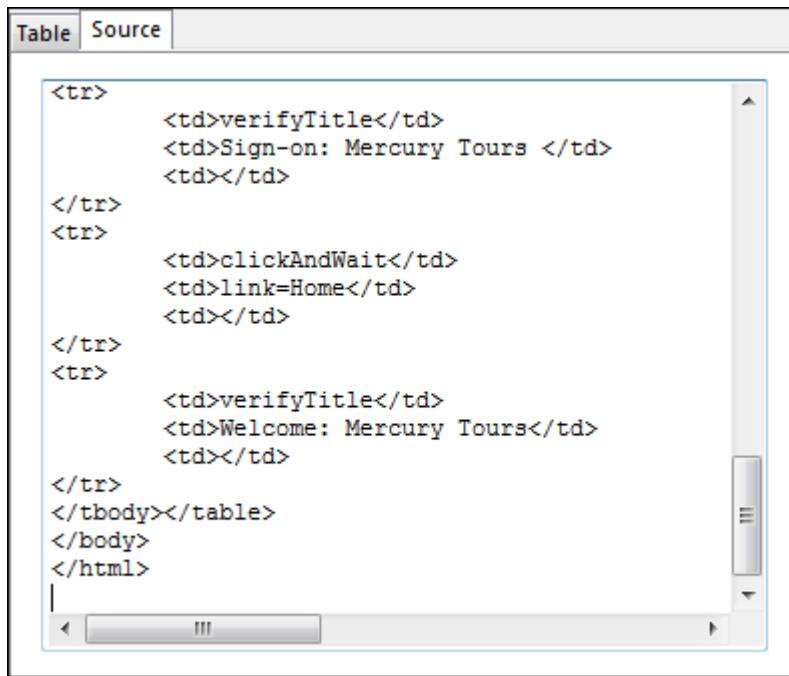


To create steps, type the name of the command in the “Command” text box.

It displays a dropdown list of commands that match with the entry that you are currently typing.

Target is any parameter (like username , password) for a command and Value is the input value (like tom,123pass) for those Targets.

## Source View



The screenshot shows the Selenium IDE interface with the 'Source' tab selected. The main pane displays an HTML script with several `<tr>` and `<td>` tags. The script includes steps like 'verifyTitle', 'Sign-on: Mercury Tours', 'clickAndWait', 'link=Home', and 'verifyTitle' again, followed by closing tags for `</tr>`, `</tbody>`, `</body>`, and `</html>`. The code is presented in a standard text editor style with syntax highlighting.

```

<tr>
    <td>verifyTitle</td>
    <td>Sign-on: Mercury Tours </td>
    <td></td>
</tr>
<tr>
    <td>clickAndWait</td>
    <td>link=Home</td>
    <td></td>
</tr>
<tr>
    <td>verifyTitle</td>
    <td>Welcome: Mercury Tours</td>
    <td></td>
</tr>
</tbody></table>
</body>
</html>

```

It displays the steps in HTML (default) format.

It also allows you to edit your script just like in the Table View.

## Log Pane

The Log Pane displays runtime messages during execution. It provides real-time updates as to what Selenium IDE is doing.

Logs are categorized into four types:

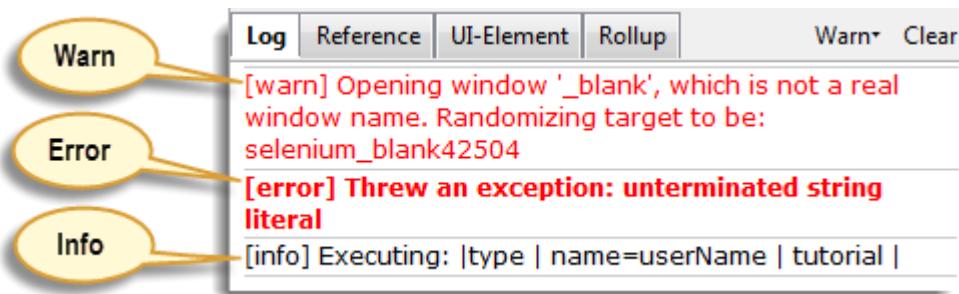
**Debug** – By default, Debug messages are not displayed in the log panel. They show up only when you filter them. They provide technical information about what Selenium IDE is doing behind the scenes. It may display messages such as a specific module has done loading, a certain function is called, or an external JavaScript file was loaded as an extension.

**Info** – It says which command Selenium IDE is currently executing.

**Warn** – These are warning messages that are encountered in special situations.

**Error – These are error messages generated when Selenium IDE fails to execute a command, or if a condition specified by “verify” or “assert” command is not met.**

TOPS Technologies



Logs can be filtered by type. For example, if you choose to select the “Error” option from the dropdown list, the Log Pane will show error messages only.

The diagram illustrates the filtering process for logs:

**Large Window (Top):**

- Log pane showing multiple log entries.
- Filter dropdown set to "Error".
- Only the [error] log entry is visible: "Actual value 'Sign-on: Mercury Tours' did not match 'Sign-on: Mercury Tours'".

**Small Window (Bottom):**

- Log pane showing the same log entry as the large window.
- Filter dropdown set to "Error".
- Only the [error] log entry is visible: "Actual value 'Sign-on: Mercury Tours' did not match 'Sign-on: Mercury Tours'".

## Reference Pane

The Reference Pane shows a concise description of the currently selected Selenese command in the Editor. It also shows the description about the locator and value to be used on that command.

A screenshot of the Selenium IDE interface focusing on the Reference pane. At the top, there's a table with three columns: Command, Target, and Value. The rows show the following commands:

Command	Target	Value
open	/	
type	userN <del>ame</del>	tutorial
type	password	tutorial
clickAndWait	login	

An orange arrow points from the 'type' command in the table to the corresponding documentation in the pane below. The pane has tabs at the top: Log, Reference (which is selected), UI-Element, and Rollup. The 'Reference' tab contains the following information:

**type(locator, value)**

Arguments:

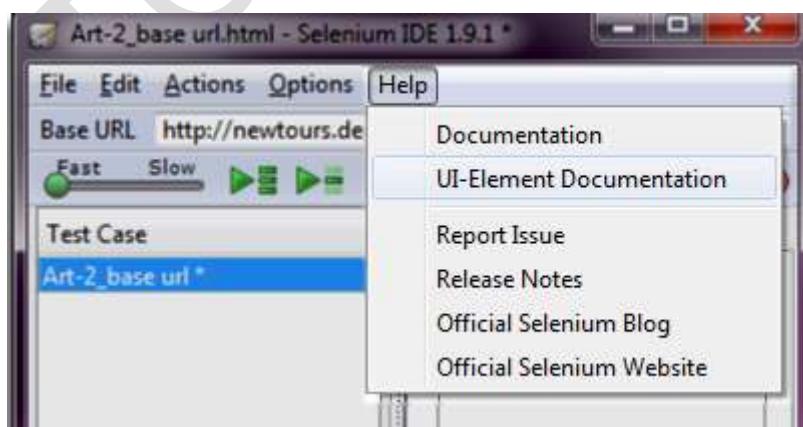
- locator - an element locator
- value - the value to type

Sets the value of an input field, as though you typed it in.

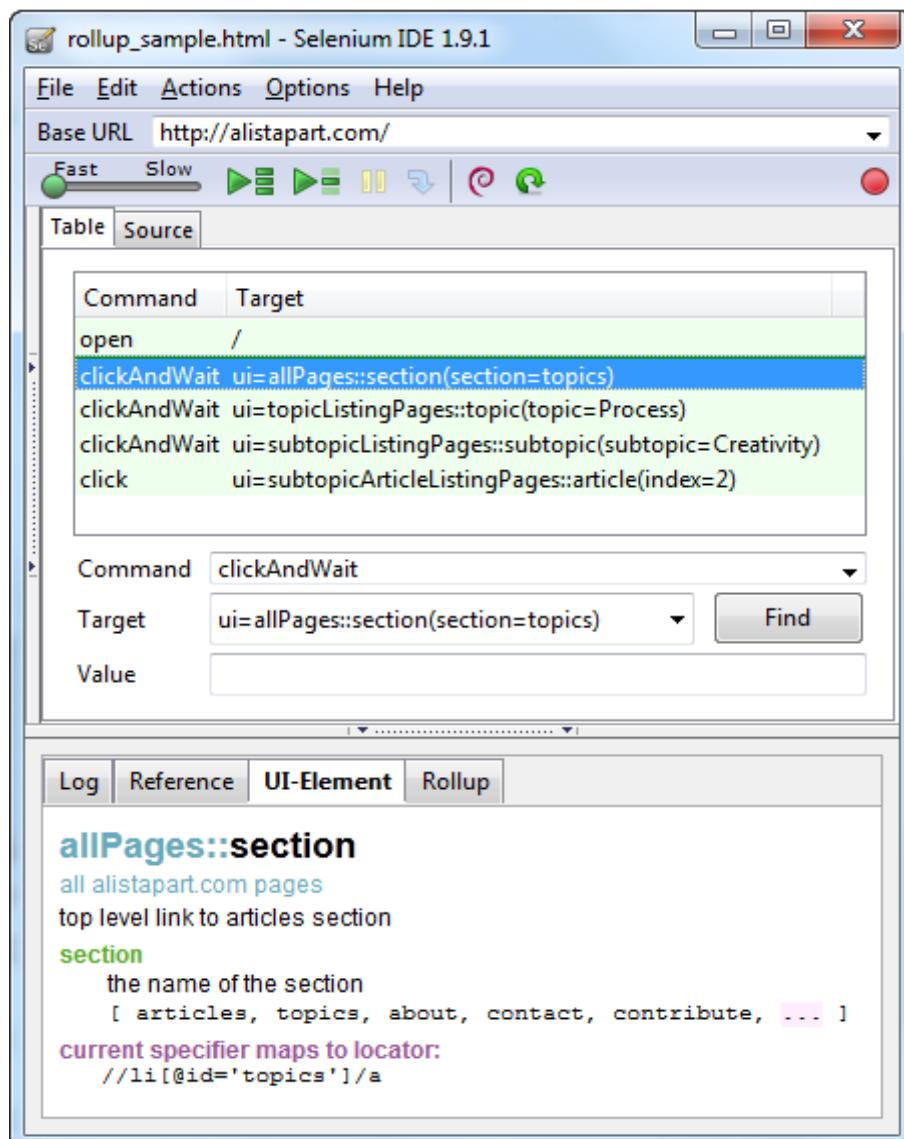
Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text.

## UI-Element Pane

The UI-Element is for advanced Selenium users. It uses JavaScript Object Notation (JSON) to define element mappings. The documentation and resources are found in the “UI Element Documentation” option under the Help menu of Selenium IDE.



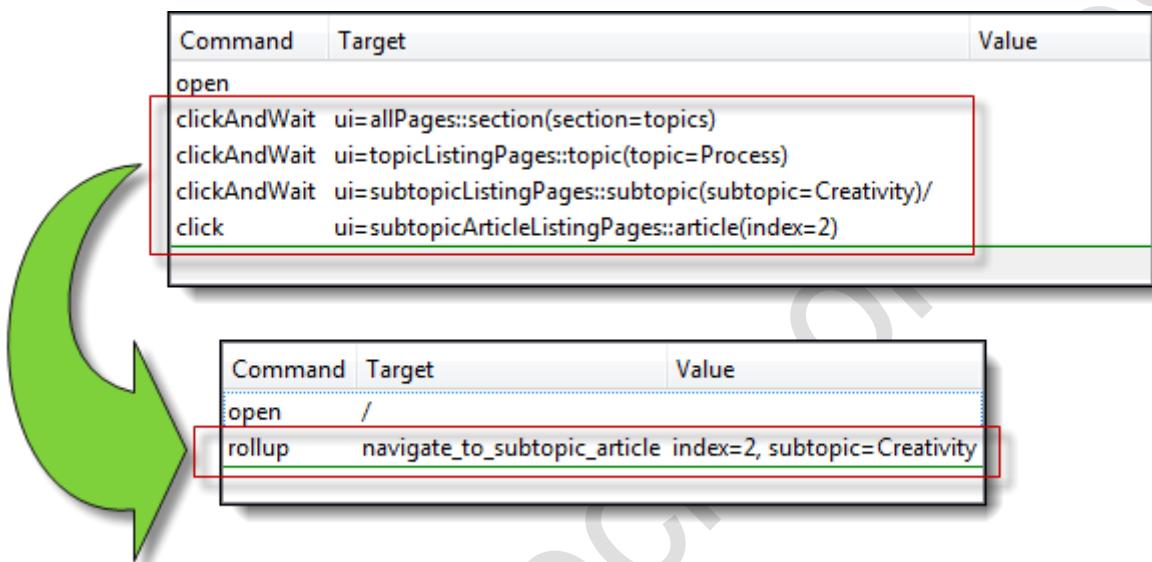
An example of a UI-element screen is shown below.



## Rollup Pane

**Rollup** allows you to execute a group of commands in one step. A group of commands is simply called as a “rollup.” It employs heavy use of JavaScript and UI-Element concepts to formulate a collection of commands that is similar to a “function” in programming languages.

Rollups are reusable; meaning, they can be used multiple times within the test case. Since rollups are groups of commands condensed into one, they contribute a lot in shortening your test script.



An example of how the contents of the rollup tab look like is shown below.

The screenshot shows the Selenium IDE interface with the 'rollup\_sample.html' file open. The 'Actions' tab is selected, displaying a table of recorded commands:

Command	Target	Value
open	/	
rollup	navigate_to_subtopic_article	index=2, subtopic=Creativity

Below the table, there are three input fields for modifying the selected command:

- Command: rollup
- Target: navigate\_to\_subtopic\_article
- Value: index=2, subtopic=Creativity

At the bottom of the window, the 'Rollup' tab is selected, showing the detailed documentation for the 'navigate\_to\_subtopic\_article' command:

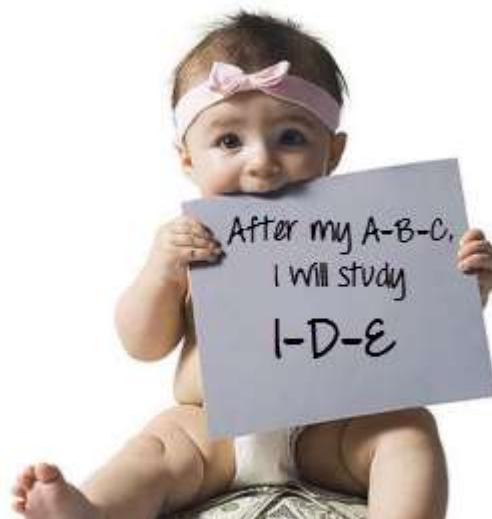
**navigate\_to\_subtopic\_article**  
navigate to an article listed under a subtopic.  
preconditions: current page contains the section menu (most pages should)  
postconditions: navigated to an article page  
**subtopic**  
the subtopic whose article listing to navigate to  
[ Browsers, CSS, Flash, HTML and XHTML, Scripting, ... ]

## Summary

**Selenium IDE (Integrated Development Environment) is the simplest tool in the Selenium Suite.**

**It must only be used as a prototyping tool.**

**Knowledge of JavaScript and HTML is required for intermediate topics such as executing the “runScript” and “rollup” commands. A rollup is a collection of commands that you can reuse to shorten your test scripts significantly. Locators are identifiers that tell Selenium IDE how to access an element.**



**Firebug (or any similar add-on) is used to obtain locator values.**

**The menu bar is used in creating, modifying, and exporting test cases into formats useable by Selenium RC and WebDriver.**

**The default format for Selenese commands is HTML.**

**The “Options” menu provides access to various configurations for Selenium IDE.**

**The Base URL is useful in accessing relative URLs.**

**The Test Case Pane shows the list of currently opened test cases and a concise summary of test runs.**

**The Editor provides the interface for your test scripts.**

**The Table View shows your script in tabular format with “Command”, “Target”, and “Value” as the columns.**

**The Source View shows your script in HTML format.**

when executing tests.

The UI-Element and Rollup tabs are for advanced Selenium IDE users only. They both require considerable effort in coding JavaScript.

UI-Element allows you to conveniently map UI elements using JavaScript Object Notation (JSON)

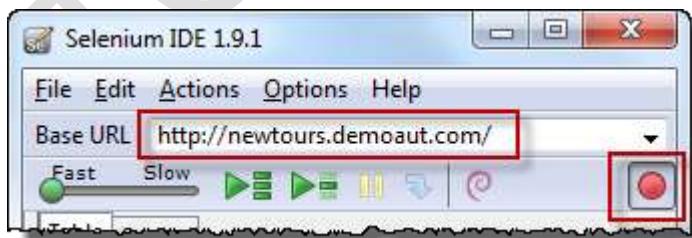
## First Selenium Test Script

### Create a Script by Recording

#### Step 1 Launch Firefox and Selenium IDE.

Type the value for our Base URL: <http://newtours.demoaut.com/>.

Toggle the Record button on (if it is not yet toggled on by default).



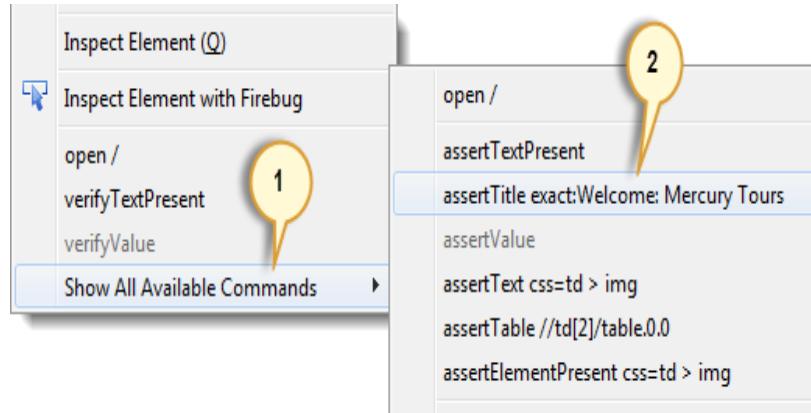
**Step 2** In Firefox, navigate to <http://newtours.demoaut.com/>. Firefox should take you to the page similar to the one shown below.

The screenshot shows the homepage of Mercury Tours. At the top, there's a banner with the text "one cool summer" and the Aruba logo. Below the banner, there are navigation links: SIGN-ON, REGISTER, SUPPORT, and CONTACT. On the left side, there's a sidebar with links for Home, Flights, Hotels, Car Rentals, Cruises, Destinations, and Vacations. Below this, there's a section for the "HTML VERSION" and a "SAVINGS!" offer for renting a car. The main content area features a "Featured Destination" section for ARUBA, which includes a globe icon and a photo of a beach. A descriptive text about Aruba follows: "This island is surrounded by coral reefs, offers guaranteed sunshine and is blessed with beautiful beaches. Luxury resorts have taken up residence along most of the beachfronts on the southern coast, but there are still undeveloped areas on the exposed northern coast, and much of the interior is inhabited by nothing more substantial than goats." Below this, there's a "Specials" section listing flight deals: Atlanta to Las Vegas (\$398), Boston to San Francisco (\$513), and Los Angeles to Chicago (\$168). To the right, there's a sign-in form with fields for User Name and Password, and a "Sign-In" button. There are also links for Destinations, Vacations, Register, and a link to register for the Mercury program.

**Step 3** Right-click on any blank space within the page, like on the Mercury Tours logo on the upper left corner. This will bring up the Selenium IDE context menu. Note: Do not click on any hyperlinked objects or images

Select the “Show Available Commands” option.

Then, select “assertTitle exact:Welcome: Mercury Tours”. This is a command that makes sure that the page title is correct.

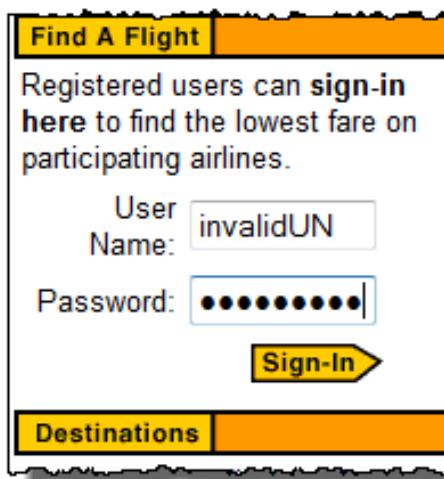


Comm...	Target	Value
open	/	
assertTitle exact:Welcome:	Mercury Tours	

After clicking on the assertTitle context menu option, your Selenium IDE Editor pane should now contain the following commands

**Step 4** In the “User Name” text box of Mercury Tours, type an invalid username, “invalidUN”.

In the “Password” text box, type an invalid password, “invalidPW”.



Comm...	Target	Value
open	/	
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW

Your Editor should now look like this

TOPS Technologies

## Step 5

Click on the “Sign-In” button. Firefox should take you to this page.

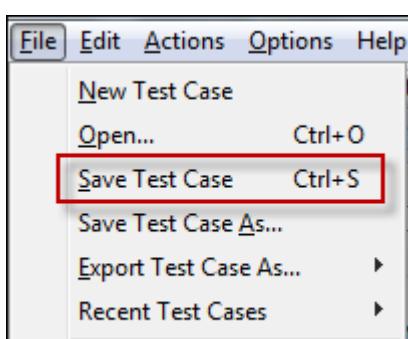
## Step 6

Toggle the record button off to stop recording. Your script should now look like the one shown below.

Command	Target	Value
open		
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndWait	name=login	

## Step 7

Now that we are done with our test script, we shall save it in a test case. In the File menu, select “Save Test Case”. Alternatively, you can simply press Ctrl+S.



## Step 8

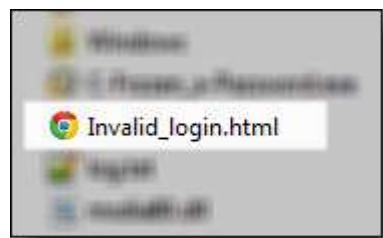
Choose your desired location, and then name the test case as “Invalid\_login”.

Click the “Save” button.



## Step 9.

Notice that the file was saved as HTML.



## Step 10.

Go back to Selenium IDE and click the Playback button to execute the whole script. Selenium IDE should be able to replicate everything flawlessly.

Test Case	Table	Source																		
Invalid_login	<table border="1"> <thead> <tr> <th>Command</th> <th>Target</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>open</td> <td></td> <td></td> </tr> <tr> <td>assertTitle</td> <td>exact:Welcome: Mercury Tours</td> <td></td> </tr> <tr> <td>type</td> <td>name=userName</td> <td>invalidUN</td> </tr> <tr> <td>type</td> <td>name=password</td> <td>invalidPW</td> </tr> <tr> <td>clickAndWait</td> <td>name=login</td> <td></td> </tr> </tbody> </table>	Command	Target	Value	open			assertTitle	exact:Welcome: Mercury Tours		type	name=userName	invalidUN	type	name=password	invalidPW	clickAndWait	name=login		
Command	Target	Value																		
open																				
assertTitle	exact:Welcome: Mercury Tours																			
type	name=userName	invalidUN																		
type	name=password	invalidPW																		
clickAndWait	name=login																			
		<p>Runs: 1</p> <p>Failures: 0</p>																		

# Introduction to Selenium Commands – Selenese

Selenese commands can have up to a maximum of two parameters: target and value.

Parameters are not required all the time. It depends on how many the command will need.

For a complete reference of Selenese commands, click [here](#)

## 3 Types of Commands

<b>Actions</b>	<p>These are commands that directly interact with page elements.</p> <p>Example: the "click" command is an action because you directly interact with the element you are clicking at.</p> <p>The "type" command is also an action because you are putting values into a text box, and the text box shows them to you in return. There is a two-way interaction between you and the text box.</p>
<b>Accessors</b>	<p>They are commands that allow you to store values to a variable.</p> <p>Example: the "storeTitle" command is an accessor because it only "reads" the page title and saves it in a variable. It does not interact with any element on the page.</p>
<b>Assertions</b>	<p>They are commands that verify if a certain condition is met.</p> <p><b>3 Types of Assertions</b></p> <ul style="list-style-type: none"> <li>▪ <b>Assert.</b> When an "assert" command fails, the test is stopped immediately.</li> <li>▪ <b>Verify.</b> When a "verify" command fails, Selenium IDE logs this failure and continues with the test execution.</li> <li>▪ <b>WaitFor.</b> Before proceeding to the next command, "waitFor" commands will first wait for a certain condition to become true. <ul style="list-style-type: none"> <li>▪ If the condition becomes true within the waiting period, the step passes.</li> <li>▪ If the condition does not become true, the step fails. Failure is logged, and test execution proceeds to the next command.</li> <li>▪ By default, timeout value is set to 30 seconds. You can change this in the Selenium IDE Options dialog under the General tab.</li> </ul> </li> </ul>

## Assert vs. Verify

**ASSERT**

test execution  
was halted in  
this part

Command	Target	Value
open		
assertTitle	Welcome: Venus Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndW...	name=login	

Command:   
Target:  Find  
Value:

Log | Reference | UI-Element | Rollup | Info | Clear

[info] Executing: |open |||  
[info] Executing: |assertTitle | Welcome: Venus Tours ||  
**[error] Actual value 'Welcome: Mercury Tours'  
did not match 'Welcome: Venus Tours'**

**Common Commands**

Command	Number of Parameters	Description
open	0 - 2	Opens a page using a URL.
click/clickAndWait	1	Clicks on a specified element.
type/typeKeys	2	Types a sequence of characters.
verifyTitle/assertTitle	1	Compares the actual page title with an expected value.
verifyTextPresent	1	Checks if a certain text is found within the page.
verifyElementPresent	1	Checks the presence of a certain element.
verifyTable	2	Compares the contents of a table with expected values.
waitForPageToLoad	1	Pauses execution until the page is loaded completely.
waitForElementPresent	1	Pauses execution until the specified element becomes present.

# Create a Script Manually with Firebug

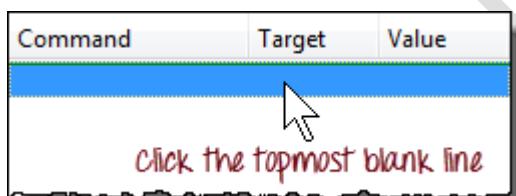
## Step 1 Open Firefox and Selenium IDE.

Type the base URL (<http://newtours.demoaut.com/>).

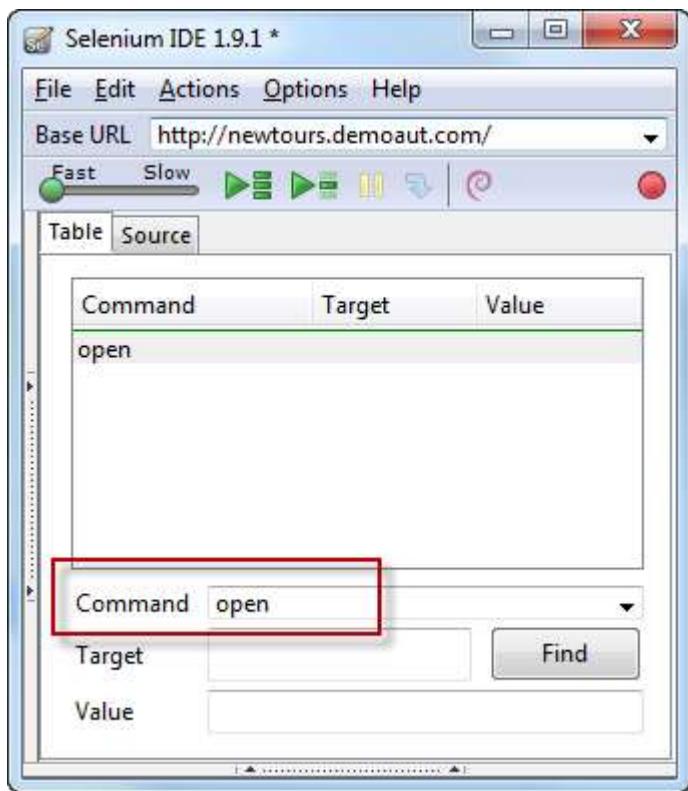
The record button should be OFF.



## Step 2 Click on the topmost blank line in the Editor.



Type “open” in the Command text box and press Enter.



### Step 3 Navigate Firefox to our base URL and activate Firebug

In the Selenium IDE Editor pane, select the second line (the line below the “open” command) and create the second command by typing “assertTitle” on the Command box.

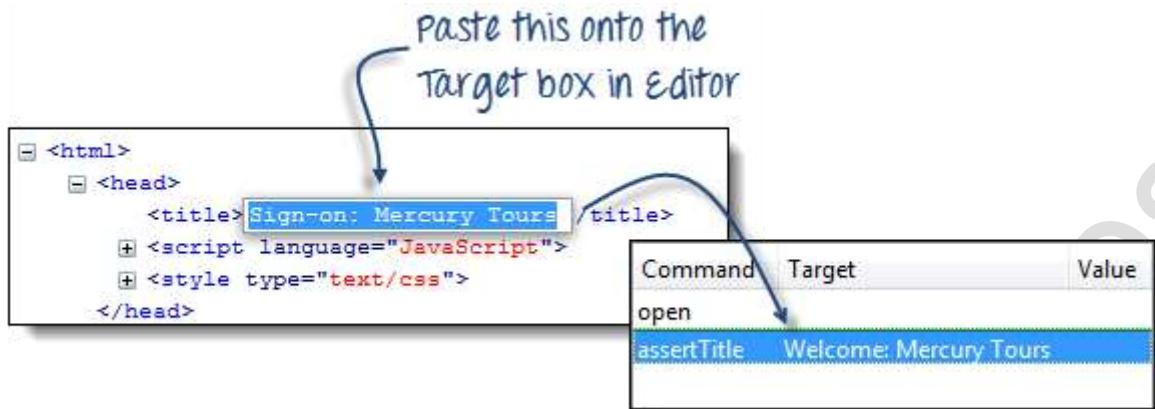
Feel free to use the autocomplete feature.

The image contains two side-by-side screenshots of the Selenium IDE Editor pane. In the left screenshot, the 'Command' field contains 'assertT'. In the right screenshot, the 'Command' field contains 'assertTitle', and the dropdown menu below it shows 'assertTitle' highlighted with a red arrow pointing to it. Both screenshots show the 'Command' dropdown menu open with several options listed.

## Step 4

In Firebug, expand the <head> tag to display the <title> tag.

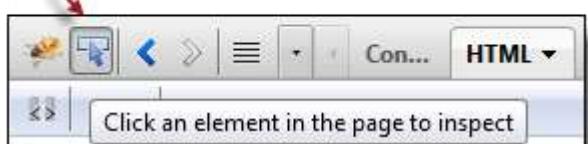
Click on the value of the <title> tag (which is “Welcome: Mercury Tours”) and paste it onto the Target field in the Editor.



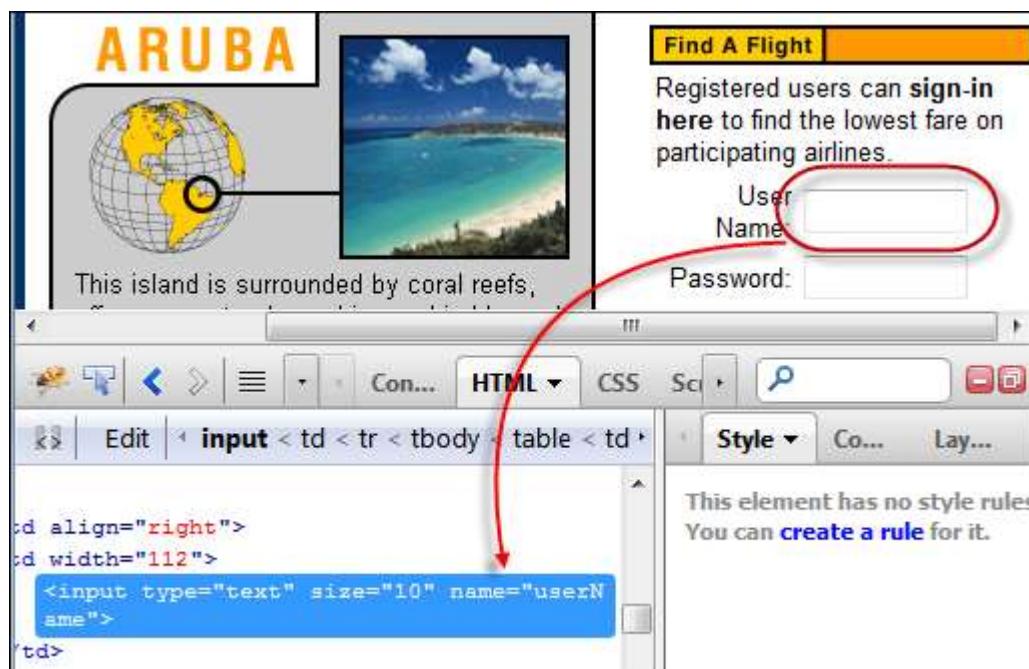
## Step 5

To create the third command, click on the third blank line in the Editor and key-in “type” on the Command text box.

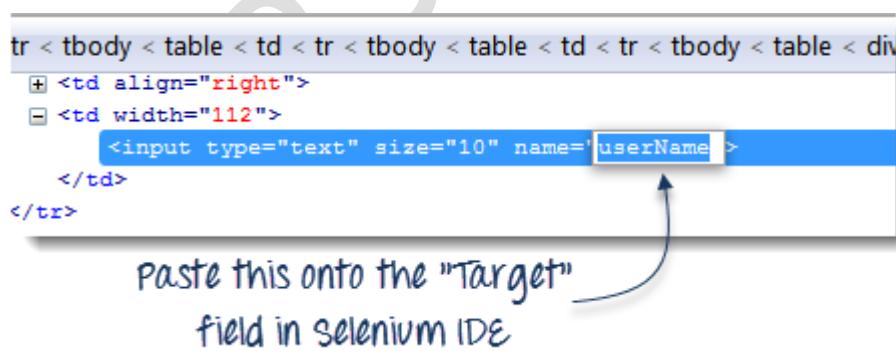
In Firebug, click on the “Inspect” button.



Click on the User Name text box. Notice that Firebug automatically shows you the HTML code for that element.



**Step 6** Notice that the User Name text box does not have an ID, but it has a NAME attribute. We shall, therefore, use its NAME as the locator. Copy the NAME value and paste it onto the Target field in Selenium IDE.



Still in the Target text box, prefix "userName" with "name=", indicating that Selenium IDE should target an element whose NAME attribute is "userName."

Command	Target	Value
open		
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	

Type “invalidUN” in the Value text box of Selenium IDE. Your test script should now look like the image below. We are done with the third command. Note: Instead of invalidUN , you may enter any other text string. But Selenium IDE is case sensitive and you type values/attributes exactly like in application.

Command	Target	Value
open		
assertTitle	Welcome: Mercury Tours	
type	name=userName	invalidUN

## Step 7

To create the fourth command, key-in “type” on the Command text box.

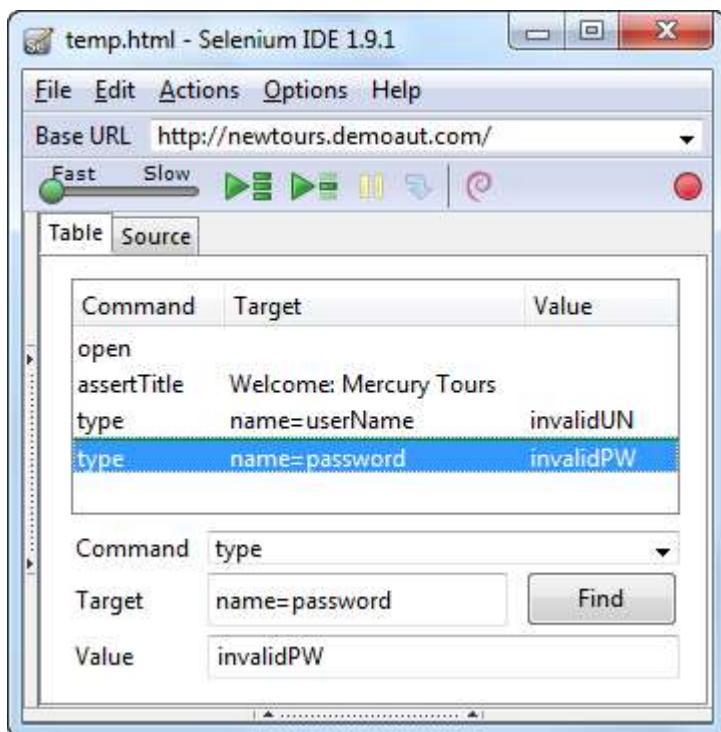
Again, use Firebug’s “Inspect” button to get the locator for the “Password” text box.

```
<tr>
  <td align="right">
    <td width="112">
      <input type="password" size="10" name="password" />
    </td>
  </tr>
<tr>
```

The Password text box only has the NAME attribute so we will use it as our locator

Paste the NAME attribute (“password”) onto the Target field and prefix it with “name=”

Type “invalidPW” in the Value field in Selenium IDE. Your test script should now look like the image below.



## Step 8

For the fifth command, type “clickAndWait” on the Command text box in Selenium IDE.

Use Firebug’s “Inspect” button to get the locator for the “Sign In” button.

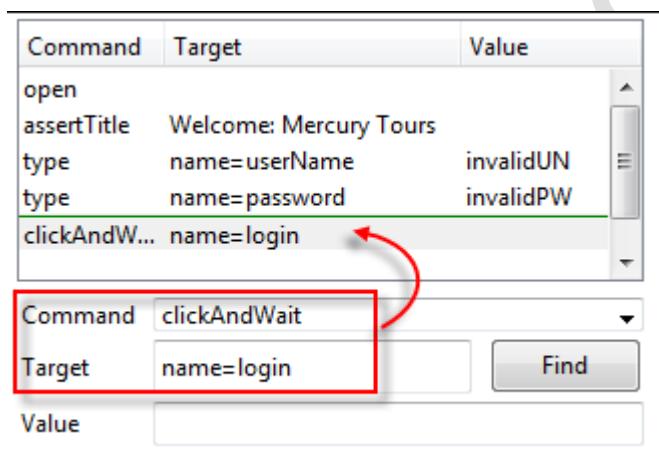


```
<td width="112">
  <div align="center">
    <input width="58" type="image" height="17" border="0"
      alt="Sign-In" src="/images/btn_signin.gif" value="Login" name="login" />
  </div>
</td>
```

Again, the only available locator is the NAME attribute so this will be the one that we shall use.

Paste the value of the NAME attribute (“login”) onto the Target text box and prefix it with “name=”.

Your test script should now look like the image below.



Command	Target	Value
open		
assertTitle	Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndW...	name=login	

Command	Target	Value
clickAndWait	name=login	

## Step 9

Save the test case in the same way as we did in the previous section.

### Using the Find Button

The Find button in Selenium IDE is used to verify if what we had put in the Target text box is indeed the correct UI element.

Let us use the **Invalid\_login** test case that we created in the previous sections. Click on any command with a Target entry, say, the third command.

Command	Target	Value
open		
assertTitle	Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndW...	name=login	

**Bottom Panel:**

Command: type  
Target: name=userName  
Value: invalidUN

Click on the Find button. Notice that the User Name text box within the Mercury Tours page becomes highlighted for a second.

**Find A Flight**

Registered users can sign-in here to find the lowest fare on participating airlines.

User Name:

Password:

Sign-In

Destinations

Every time the Find button is clicked, this element becomes highlighted with yellow and bordered with light green.

This indicates that Selenium IDE was able to detect and access the expected element correctly. If the Find button highlighted a different element or no element at all, then there must be something wrong with your script.

TOPS Technologies

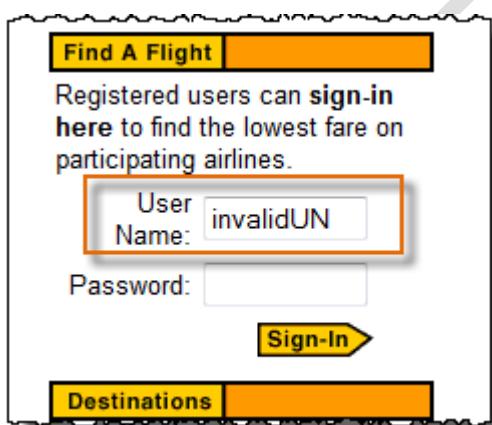
# Execute Command

**Step 1.** Make sure that your browser is on the Mercury Tours homepage. Click on the command you wish to execute. In this example, click on the “type | userName | invalidUN” line.

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	

**Step 2.** Press “X” on your keyboard.

**Step 3.** Observe that the text box for username becomes populated with the text “invalidUN”



## Start point

A start point is an indicator that tells Selenium IDE which line the execution will start. Its shortcut key is “S”.

Symbol of a  
“start point”



Command	Target	Value
open	/	
type	userName	invalidUN
▶ type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

In the example above, playback will start on the third line (type | password | invalidPW). You can only have one start point in a single test script.

Start point is similar to Execute Command in such that they are dependent on the currently displayed page. The start point will fail if you are on the wrong page.

## Breakpoints

Breakpoints are indicators that tell Selenium IDE where to automatically pause the test. The shortcut key is “B”.



Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

The yellow highlight means that the current step is pending. This proves that Selenium IDE has paused execution on that step. You can have multiple breakpoints in one test case.

## Step

It allows you to execute succeeding commands one at a time after pausing the test case. Let us use the scenario in the previous section “Breakpoints.”

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

Before clicking “Step.”

The test case pauses at the line “clickAndWait | login”.

After clicking “Step.”

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

The “clickAndWait | login” line is run and pauses to the next command (verifyTitle | Sign-on: Mercury Tours).

Notice that the next line is paused even though there is no breakpoint there. This is the main purpose of the Step feature – it executes the succeeding commands one at a time to give you more time to inspect the outcome after each step.

## Important Things to Note When Using Other Formats in Source View

**Selenium IDE works well only with HTML – other formats are still in experimental mode. It is NOT advisable to create or edit tests using other formats in Source View because there is still a lot of work needed to make it stable. Below are the known bugs as of version 1.9.1.**

**You will not be able to perform playback nor switch back to Table View unless you revert to HTML.**

**The only way to add commands safely on the source code is by recording them.**

**When you modify the source code manually, all of it will be lost when you switch to another format.**

**Though you can save your test case while in Source View, Selenium IDE will not be able to open it.**

**The recommended way to convert Selenese tests is to use the “Export Test Case As...” option under the File menu, and not through the Source View..**

## Summary

- Test scripts can be created either by recording or typing the commands and parameters manually.
- When creating scripts manually, Firebug is used to get the locator.
- The Find button is used to check that the command is able to access the correct element.
- Table View displays a test script in tabular form while Source View displays it in HTML format.
- Changing the Source View to a non-HTML format is still experimental.
- Do not use the Source View in creating tests in other formats. Use the Export features instead.

Parameters are not required all the time. It depends upon the command.

# Given Below Are Some Of The More Popular And Common Myths About Software Testing.

<b>1</b>	<b>Myths: Testing is too expensive.</b>
	<b>Reality:</b> There is a saying, pay less for testing during software development or pay more for maintenance or correction later. Early testing saves both time and cost in many aspects however, reducing the cost without testing may result in the improper design of a software application rendering the product useless.
<b>2</b>	<b>Myths: Testing is time consuming.</b>
	<b>Reality:</b> During the SDLC phases testing is never a time consuming process. However diagnosing and fixing the error which is identified during proper testing is a time consuming but productive activity.
<b>3</b>	<b>Myths: Testing cannot be started if the product is not fully developed.</b>
	<b>Reality:</b> No doubt, testing depends on the source code but reviewing requirements and developing test cases is independent from the developed code. However iterative or incremental approach as a development life cycle model may reduce the dependency of testing on the fully developed software.
<b>4</b>	<b>Myths: Complete Testing is Possible.</b>
	<b>Reality:</b> It becomes an issue when a client or tester thinks that complete testing is possible. It is possible that all paths have been tested by the team but occurrence of complete testing is never possible. There might be some scenarios that are never executed by the test team or the client during the software development life cycle and may be executed once the project has been deployed.
<b>5</b>	<b>Myths: If the software is tested then it must be bug free.</b>
	<b>Reality:</b> This is a very common myth which clients, Project Managers and the management team believe in. No one can say with absolute certainty that a software application is 100% bug free even if a tester with superb testing skills has tested the application.
<b>6</b>	<b>Myths: Missed defects are due to Testers.</b>
	<b>Reality:</b> It is not a correct approach to blame testers for bugs that remain in the application even after testing has been performed. This myth relates to Time, Cost, and Requirements changing Constraints. However the test strategy may also result in bugs being missed by the testing team.
<b>7</b>	<b>Myths: Testers should be responsible for the quality of a product.</b>
	<b>Reality:</b> It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Tester's responsibilities include the identification of bugs to the stakeholders and then it is their decision whether they will fix the bug or release the software. Releasing the software at the time puts more pressure on the testers as they will be blamed for any error.

8	<b>Myths: Test Automation should be used wherever it is possible to use it and to reduce time.</b>
	<p><b>Reality:</b> Yes it is true that Test Automation reduces the testing time but it is not possible to start Test Automation at any time during Software development. Test Automaton should be started when the software has been manually tested and is stable to some extent. Moreover, Test Automation can never be used if requirements keep changing.</p>
9	<b>Myths: Any one can test a Software application.</b>
	<p><b>Reality:</b> People outside the IT industry think and even believe that any one can test the software and testing is not a creative job. However testers know very well that this is myth. Thinking alternatives scenarios, try to crash the Software with the intent to explore potential bugs is not possible for the person who developed it.</p>
10	<b>Myths: A tester's task is only to find bugs.</b>
	<p><b>Reality:</b> Finding bugs in the Software is the task of testers but at the same time they are domain experts of the particular software. Developers are only responsible for the specific component or area that is assigned to them but testers understand the overall workings of the software, what the dependencies are and what the impacts of one module on another module are.</p>

## Testing Definitions

- 1. Acceptance Testing:** Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.  
It is usually performed by the customer.
- 2. Accessibility Testing:** Type of testing which determines the usability of a product to the people having disabilities (deaf, blind, mentally disabled etc). The evaluation process is conducted by persons having disabilities.
- 3. Active Testing:** Type of testing consisting in introducing test data and analyzing the execution results. It is usually conducted by the testing teams.
- 4. Agile Testing:** Software testing practice that follows the principles of the agile manifesto, emphasizing testing from the perspective of customers who will utilize the system. It is usually performed by the QA teams.
- 5. Age Testing:** Type of testing which evaluates a system's ability to perform in the future. The evaluation process is conducted by testing teams.

**6. Ad-hoc Testing:** Testing performed without planning and documentation - the tester tries to 'break' the system by randomly trying the system's functionality. It is performed by the testing teams.

**7. Alpha Testing:** Type of testing a software product or system conducted at the developer's site. Usually it is performed by the end user.

**8. Assertion Testing:** Type of testing consisting in verifying if the conditions confirm the product requirements. It is performed by the testing teams.

**9. API Testing:** Testing technique similar to unit testing in that it targets the code level. API Testing differs from unit testing in that it is typically a QA task and not a developer task.

**10. All-pairs Testing:** Combinatorial testing method that tests all possible discrete combinations of input parameters. It is performed by the testing teams.

**11. Automated Testing:** Testing technique that uses automation testing tools to control the environment set-up, test execution and results reporting. It is performed by a computer and is used inside the testing teams.

**12. Basis Path Testing:** A testing mechanism which derives a logical complexity measure of a procedural design and use this as a guide for defining a basic set of execution paths. It is used by testing teams when defining test cases.

**13. Backward Compatibility Testing:** Testing method which verifies the behavior of the developed software with older versions of the test environment. It is performed by testing teams.

**14. Beta Testing:** Final testing before releasing application for commercial purpose. It is typically done by end-users or others.

**15. Benchmark Testing:** Testing technique that uses representative sets of programs and data designed to evaluate the performance of computer hardware and software in a given configuration.  
It is performed by testing teams.

**16. Big Bang Integration Testing:** Testing technique which integrates individual program modules only when everything is ready. It is performed by the testing teams.

**17. Binary Portability Testing:** Technique that tests an executable application for portability across system platforms and environments, usually for conformation to an ABI specification.  
It is performed by the testing teams.

**18. Boundary Value Testing:** Software testing technique in which tests are designed to include representatives of boundary values. It is performed by the QA testing teams.

**19. Bottom Up Integration Testing:** In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time. It is usually performed by the testing teams.

**20. Branch Testing:** Testing technique in which all branches in the program source code are tested at least once. This is done by the developer.

**21. Breadth Testing:** A test suite that exercises the full functionality of a product but does not test features in detail. It is performed by testing teams.

**22. Black box Testing:** A method of software testing that verifies the functionality of an application without having specific knowledge of the application's code/internal structure. Tests are based on requirements and functionality. It is performed by QA teams.

**23. Code-driven Testing:** Testing technique that uses testing frameworks (such as xUnit) that allow the execution of unit tests to determine whether various sections of the code are acting as expected under various circumstances. It is performed by the development teams.

**24. Compatibility Testing:** Testing technique that validates how well a software performs in a particular hardware/software/operating system/network environment. It is performed by the testing teams.

**25. Comparison Testing:** Testing technique which compares the product strengths and weaknesses with previous versions or other similar products. Can be performed by tester, developers, product managers or product owners.

**26. Component Testing:** Testing technique similar to unit testing but with a higher level of integration - testing is done in the context of the application instead of just directly testing a specific method. Can be performed by testing or development teams.

**27. Configuration Testing:** Testing technique which determines minimal and optimal configuration of hardware and software, and the effect of adding or modifying resources such as memory, disk drives and CPU. Usually it is performed by the performance testing engineers.

**28. Condition Coverage Testing:** Type of software testing where each condition is executed by making it true and false, in each of the ways at least once. It is typically made by the automation testing teams.

**29. Compliance Testing:** Type of testing which checks whether the system was developed in accordance with standards, procedures and guidelines. It is usually performed by external companies which offer "Certified OGC Compliant" brand.

**30. Concurrency Testing:** Multi-user testing geared towards determining the effects of accessing the same application code, module or database records. It is usually done by performance engineers.

**31. Conformance Testing:** The process of testing that an implementation conforms to the specification on which it is based. It is usually performed by testing teams.

**32. Context Driven Testing:** An Agile Testing technique that advocates continuous and creative

evaluation of testing opportunities in light of the potential information revealed and the value of that information to the organization at a specific moment. It is usually performed by Agile testing teams.

**33. Conversion Testing:** Testing of programs or procedures used to convert data from existing systems for use in replacement systems. It is usually performed by the QA teams.

**34. Decision Coverage Testing:** Type of software testing where each condition/decision is executed by setting it on true/false. It is typically made by the automation testing teams.

**35. Destructive Testing:** Type of testing in which the tests are carried out to the specimen's failure, in order to understand a specimen's structural performance or material behaviour under different loads. It is usually performed by QA teams.

**36. Dependency Testing:** Testing type which examines an application's requirements for pre-existing software, initial states and configuration in order to maintain proper functionality. It is usually performed by testing teams.

**37. Dynamic Testing:** Term used in software engineering to describe the testing of the dynamic behavior of code. It is typically performed by testing teams.

**38. Domain Testing:** White box testing technique which contains checkings that the program accepts only valid input. It is usually done by software development teams and occasionally by automation testing teams.

**39. Error-Handling Testing:** Software testing type which determines the ability of the system to properly process erroneous transactions. It is usually performed by the testing teams.

**40. End-to-end Testing:** Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. It is performed by QA teams.

**41. Endurance Testing:** Type of testing which checks for memory leaks or other problems that may occur with prolonged execution. It is usually performed by performance engineers.

**42. Exploratory Testing:** Black box testing technique performed without planning and documentation. It is usually performed by manual testers.

**43. Equivalence Partitioning Testing:** Software testing technique that divides the input data of a software unit into partitions of data from which test cases can be derived. It is usually performed by the QA teams.

**44. Fault injection Testing:** Element of a comprehensive test strategy that enables the tester to concentrate on the manner in which the application under test is able to handle exceptions. It is performed by QA teams.

**45. Formal verification Testing:** The act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. It is usually performed by QA teams.

**46. Functional Testing:** Type of black box testing that bases its test cases on the specifications of the software component under test. It is performed by testing teams.

**47. Fuzz Testing:** Software testing technique that provides invalid, unexpected, or random data to the inputs of a program - a special area of mutation testing. Fuzz testing is performed by testing teams.

**48. Gorilla Testing:** Software testing technique which focuses on heavily testing of one particular module. It is performed by quality assurance teams, usually when running full testing.

**49. Gray Box Testing:** A combination of Black Box and White Box testing methodologies: testing a piece of software against its specification but using some knowledge of its internal workings. It can be performed by either development or testing teams.

**50. Glass box Testing:** Similar to white box testing, based on knowledge of the internal logic of an application's code. It is performed by development teams.

**51. GUI software Testing:** The process of testing a product that uses a graphical user interface, to ensure it meets its written specifications. This is normally done by the testing teams.

**52. Globalization Testing:** Testing method that checks proper functionality of the product with any of the culture/locale settings using every type of international input possible. It is performed by the testing team.

**53. Hybrid Integration Testing:** Testing technique which combines top-down and bottom-up integration techniques in order leverage benefits of these kind of testing. It is usually performed by the testing teams.

**54. Integration Testing:** The phase in software testing in which individual software modules are combined and tested as a group. It is usually conducted by testing teams.

**55. Interface Testing:** Testing conducted to evaluate whether systems or components pass data and control correctly to one another. It is usually performed by both testing and development teams.

**56. Install/uninstall Testing:** Quality assurance work that focuses on what customers will need to do to install and set up the new software successfully. It may involve full, partial or upgrades install/uninstall processes and is typically done by the software testing engineer in conjunction with the configuration manager.

**57. Internationalization Testing:** The process which ensures that product's functionality is not broken and all the messages are properly externalized when used in different languages and locale. It is usually performed by the testing teams.

**58. Inter-Systems Testing:** Testing technique that focuses on testing the application to ensure that interconnection between application functions correctly. It is usually done by the testing teams.

**59. Keyword-driven Testing:** Also known as table-driven testing or action-word testing, is a software testing methodology for automated testing that separates the test creation process into two distinct stages: a Planning Stage and an

Implementation Stage. It can be used by either manual or automation testing teams.

**60. Load Testing:** Testing technique that puts demand on a system or device and measures its response. It is usually conducted by the performance engineers.

**61. Localization Testing:** Part of software testing process focused on adapting a globalized application to a particular culture/locale. It is normally done by the testing teams.

**62. Loop Testing:** A white box testing technique that exercises program loops. It is performed by the development teams.

**63. Manual Scripted Testing:** Testing method in which the test cases are designed and reviewed by the team before executing it. It is done by manual testing teams.

**64. Manual-Support Testing:** Testing technique that involves testing of all the functions performed by the people while preparing the data and using these data from automated system. It is conducted by testing teams.

**65. Model-Based Testing:** The application of Model based design for designing and executing the necessary artifacts to perform software testing. It is usually performed by testing teams.

**66. Mutation Testing:** Method of software testing which involves modifying programs' source code or byte code in small ways in order to test sections of the code that are seldom or never accessed during normal tests execution. It is normally conducted by testers.

**67. Modularity-driven Testing:** Software testing technique which requires the creation of small, independent scripts that represent modules, sections, and functions of the application under test. It is usually performed by the testing team.

**68. Non-functional Testing:** Testing technique which focuses on testing of a software application for its non-functional requirements. Can be conducted by the performance engineers or by manual testing teams.

**69. Negative Testing:** Also known as "test to fail" - testing method where the tests' aim is showing that a component or system does not work. It is performed by manual or automation testers.

**70. Operational Testing:** Testing technique conducted to evaluate a system or component in its operational environment. Usually it is performed by testing teams.

**71. Orthogonal array Testing:** Systematic, statistical way of testing which can be applied in user interface testing, system testing, regression testing, configuration testing and performance testing. It is performed by the testing team.

**72. Pair Testing:** Software development technique in which two team members work together at one keyboard to test the software application. One does the testing and the other analyzes or reviews the testing. This can be done between one Tester and Developer or Business Analyst or between two testers with both participants taking turns at driving the keyboard.

**73. Passive Testing:** Testing technique consisting in monitoring the results of a running system without introducing any special test data. It is performed by the testing team.

**74. Parallel Testing:** Testing technique which has the purpose to ensure that a new application which has replaced its older version has been installed and is running correctly. It is conducted by the testing team.

**75. Path Testing:** Typical white box testing which has the goal to satisfy coverage criteria for each logical path through the program. It is usually performed by the development team.

**76. Penetration Testing:** Testing method which evaluates the security of a computer system or network by simulating an attack from a malicious source. Usually they are conducted by specialized penetration testing companies.

**77. Performance Testing:** Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements. It is usually conducted by the performance engineer.

**78. Qualification Testing:** Testing against the specifications of the previous release, usually conducted by the developer for the consumer, to demonstrate that the software meets its specified requirements.

**79. Ramp Testing:** Type of testing consisting in raising an input signal continuously until the system breaks down. It may be conducted by the testing team or the performance engineer.

**80. Regression Testing:** Type of software testing that seeks to uncover software errors after changes to the program (e.g. bug fixes or new functionality) have been made, by retesting the program. It is performed by the testing teams.

**81. Recovery Testing:** Testing technique which evaluates how well a system recovers from crashes, hardware failures, or other catastrophic problems. It is performed by the testing teams.

**82. Requirements Testing:** Testing technique which validates that the requirements are correct, complete, unambiguous, and logically consistent and allows designing a necessary and sufficient set of test cases from those requirements. It is performed by QA teams.

**83. Security Testing:** A process to determine that an information system protects data and maintains functionality as intended. It can be performed by testing teams or by specialized security-testing companies.

**84. Sanity Testing:** Testing technique which determines if a new software version is performing well enough to accept it for a major testing effort. It is performed by the testing teams.

**85. Scenario Testing:** Testing activity that uses scenarios based on a hypothetical story to help a person think through a complex problem or system for a testing environment. It is performed by the testing teams.

**86. Scalability Testing:** Part of the battery of non-functional tests which tests a software

application for measuring its capability to scale up - be it the user load supported, the number of transactions, the data volume etc. It is conducted by the performance engineer.

**87. Statement Testing:** White box testing which satisfies the criterion that each statement in a program is executed at least once during program testing. It is usually performed by the development team.

**88. Static Testing:** A form of software testing where the software isn't actually used it checks mainly for the sanity of the code, algorithm, or document. It is used by the developer who wrote the code.

**89. Stability Testing:** Testing technique which attempts to determine if an application will crash. It is usually conducted by the performance engineer.

**90. Smoke Testing:** Testing technique which examines all the basic components of a software system to ensure that they work properly. Typically, smoke testing is conducted by the testing team, immediately after a software build is made .

**91. Storage Testing:** Testing type that verifies the program under test stores data files in the correct directories and that it reserves sufficient space to prevent unexpected termination resulting from lack of space. It is usually performed by the testing team.

**92. Stress Testing:** Testing technique which evaluates a system or component at or beyond the limits of its specified requirements. It is usually conducted by the performance engineer.

**93. Structural Testing:** White box testing technique which takes into account the internal structure of a system or component and ensures that each program statement performs its intended function. It is usually performed by the software developers.

**94. System Testing:** The process of testing an integrated hardware and software system to verify that the system meets its specified requirements. It is conducted by the testing teams in both development and target environment.

**95. System integration Testing:** Testing process that exercises a software system's coexistence with others. It is usually performed by the testing teams.

**96. Top down Integration Testing:** Testing technique that involves starting at the top of a system hierarchy at the user interface and using stubs to test from the top down until the entire system has been implemented. It is conducted by the testing teams.

**97. Thread Testing:** A variation of top-down testing technique where the progressive integration of components follows the implementation of subsets of the requirements. It is usually performed by the testing teams.

**98. Upgrade Testing:** Testing technique that verifies if assets created with older versions can be used properly and that user's learning is not challenged. It is performed by the testing teams.

**99. Unit Testing:** Software verification and validation method in which a programmer tests if individual units of source code are fit for use. It is usually conducted by the development team.

**100. User Interface Testing:** Type of testing which is performed to check how user-friendly the application is. It is performed by testing teams.

**101. Usability Testing:** Testing technique which verifies the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. It is usually performed by end users.

**102. Volume Testing:** Testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner. It is usually conducted by the performance engineer.

**103. Vulnerability Testing:** Type of testing which regards application security and has the purpose to prevent problems which may affect the application integrity and stability. It can be performed by the internal testing teams or outsourced to specialized companies.

**104. White box Testing:** Testing technique based on knowledge of the internal logic of an application's code and includes tests like coverage of code statements, branches, paths, conditions. It is performed by software developers.

**105. Workflow Testing:** Scripted end-to-end testing technique which duplicates specific workflows which are expected to be utilized by the end-user. It is usually conducted by testing teams.

## Interview Question

### SDLC and Manual Testing

1. What is Software testing?
2. Objectives for Software testing?
3. Does software testing guarantee Quality?
4. What is the difference between Error, Defect & Failure?
5. What activities are performed under test planning?
6. What activities are performed under test design phase?
7. Exhaustive testing is IMPOSSIBLE- justify.
8. What is Debugging?
9. What is Endurance Testing?
10. Explain Pesticide paradox
  
11. What do you understand by "Absence of error fallacy".
12. What are the disadvantages of using a Spiral Model?
13. What are driver and stub when do we use them?
14. What is a test script and test Strategy?
15. Explain STLC
16. Explain SDLC
17. What is the difference between Load and Stress testing? Support your answer with proper example.
18. Documents are baseline (BSD) - What do you understand by this statement?
19. What is the diff between debugging and testing?
  
20. State types of Verification methods known to you.
21. White box testing is same as Debugging- Comment upon its Validity.

22. What is alpha testing and beta testing under which testing level do they fall?
  23. In which phase of SDLC, the fixing of bugs is least expensive? In which phase of SDLC, the fixing of bugs is most expensive?
  24. Describe to the basic elements you put in a defect report?
  25. Why software quality assurance is important to developers?
  26. What is Sanity testing?
  27. Give examples of non-functional requirements in software testing?
  28. What is the difference between Re-testing & Regression testing?
  29. What is Smoke testing?
  30. What is soak testing? How do you perform it in real time?
  31. Which automation tool can be used for security of website from hacking and why?
  32. How you will write test cases for integration testing? Explain me with an example
  33. What is the difference between interoperability and compatibility testing with some examples?
  34. What is exactly requirement testing?
  35. What is the importance of testing in SDLC?
  36. Difference between White Box and Black Box Testing.
  37. What is Grey testing?
  38. When do we perform white box testing?
  39. What type of testing is done for physically challenged people?
  40. What is the difference between branch coverage, decision coverage and condition coverage?
  41. What is Quality Matrix in Software Testing?
  42. What are the defects of ADHOC testing and ways to overcome them?
  43. Can environmental factors can be a reason for software failure? Explain using a suitable example.
- 
44. What is your process for determining what to automate and in what order?
  45. When do you start developing your automation tests?
  46. Realizing you won't be able to test everything how do you decide what to test first and when to set up an exit criteria?
  47. What are the disadvantages of Black Box testing?
  48. What are the steps followed in Test Strategy and Test Planning?
  49. What is the difference between Volume and Load testing?
  50. What is Business acceptance testing (BAT)? How is it different from UAT?
  51. What's a 'test plan'?
  52. What's a 'test case'? give example
  53. What should be done after a bug is found?
  54. What is Verification and Validation?
  55. How does u create new test sets in TD?

## Automation

### Load Runner

1. What is load testing?
2. What is Performance testing?
3. Explain the Load testing process?
4. What are the components of LoadRunner?
5. What Component of LoadRunner would you use to record a Script?
6. What Component of LoadRunner would you use to play Back the script in multi user mode?
  7. What is a rendezvous point?
  8. What is a scenario?

9. Explain the recording mode for web Vuser script?
  10. Why do we create parameters?
  11. What is correlation? Explain the difference between automatic correlation and manual correlation?
  12. How do you find out where correlation is required? Give few examples from your projects?
  13. Where do you set automatic correlation options?
  14. How do you debug a LoadRunner script?
  15. What are the changes you can make in run-time settings?
  16. Where do you set Iteration for Vuser testing?
  17. How do you perform functional testing under load?
  18. What is Ramp up? How do you set this?
  19. What is the advantage of running the Vuser as thread?
  20. If you want to stop the execution of your script on error, how do you do that?
  21. What is the relation between Response Time and Throughput?
  22. Explain the Configuration of your systems?
  23. How do you identify the performance bottlenecks?
  24. If web server, database and Network are all fine where could be the problem?
  25. How did you find database related issues?
  26. Explain all the web recording options?
  27. What is the difference between Overlay graph and Correlate graph?
  28. How did you plan the Load? What are the Criteria?
  29. What does vuser\_init action contain?
  30. What does vuser\_end action contain?
- 
31. What is think time? How do you change the threshold?
  32. What is the difference between standard log and extended log?
  33. Explain the following functions: - lr\_debug\_message
  34. Types of Goals in Goal-Oriented Scenario

## Software Testing Question & Answer For Interview.

**Q.1 - Explain following types of Testing:-**

### **Acceptance Testing**

Test performed by users of a new or changed system in order to approve the system and go live. Cosmetic and other small changes may still be required as a result of the test, but the system is considered stable and processing data according to requirements.

### **Active Testing**

Introducing test data and analyzing the results. Contrast with "passive test".

### **Ad Hoc Testing**

Ad-hoc testing is a type of testing in which we test the product without any test cases and with a basic knowledge of the product. It can often find problems that are not caught in regular testing. Sometimes, if testing occurs very late in the development cycle, this will be

the only kind of testing that can be performed.

For Ad hoc testing one should have strong knowledge about the application.

## **Age Testing (aging)**

Evaluating a system's ability to perform in the future. To perform these tests, hardware and/or test data are modified to a future date.

## **Alpha Testing**

First of all test newly developed hardware or software in a laboratory setting. When the first round of bugs has been fixed, the product goes into beta test with actual users. For custom software, the customer may be invited into the vendor's facilities for an alpha test to ensure the client's vision has been interpreted properly by the developer.

## **Automated Testing**

Using software to test software. Automated tests may still require human intervention to monitor stages for analysis or errors.

## **Beta Testing**

Test of new or revised hardware or software that is performed by users at their facilities under normal operating conditions. Beta testing follows alpha testing. Vendors of packaged software often offer their customers the opportunity of beta testing new releases.

## **Black Box Testing**

Testing software based on output only without any knowledge of its internal code or logic. Contrast with "white box test" and "gray box test."

## **Dirty Testing**

Testing done by putting dirty data intentionally, to record the abnormal behavior of system or flashing red or yellow messages.

## **Environment Testing**

Test of new software that determines whether all transactions flow properly between input, output and storage devices.

## **Functional Testing**

Testing software based on its functional requirements. It ensures that the program physically works the way it was intended and all required menu options are present. It also ensures that the program conforms to the industry standards relevant to that environment; for example, in a Windows program, pressing F1 brings up help.

## **Fuzz Testing**

Fuzz testing is a method of testing applications by randomly altering or corrupting input data. Fuzz testing is a simple technique but it can show important defects that need addressing. Corrupt data can cause applications to crash or behave unexpectedly. Steps to perform fuzz testing is listed below.

1. Gather the correct set of input data for your application
2. Change some or all parts of the input data with random or corrupt data
3. Pass this modified input data to your application and observe what happens

## Gray Box Testing

Testing software with some knowledge of its internal code or logic. Contrast with "white box test" and "black box test."

## Negative Testing

Using invalid input to test a program's error handling.

## Passive Testing

Monitoring the results of a running system without introducing any special test data. Contrast with "active test" (above).

## Recovery Testing

Testing a system's ability to recover from a hardware or software failure.

## Regression Testing

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. It ensures that there should be no negative impact to any other functionality which was offered previously.

## Smoke Testing

Once new build comes for testing then test the main functionality and ensure that the build is stable and further testing can be possible. This is also known as a build verification testing.

## Sanity Testing

Once a new build is obtained with minor revisions, instead of doing a through regression, a sanity is performed so as to ascertain the build has indeed rectified the issues and no further issue has been introduced by the fixes. It's generally a subset of regression testing and a group of test cases are executed that are related with the changes made to the application. Sanity testing is like doing some specialized testing which is used to find problems in particular functionality

## System Testing

System testing is black box testing and performed by the test team. The purpose of system testing is to validate an application's accuracy and completeness in performing the functions designed. System testing simulates real life scenarios and tests all functions of the system that are required in real life.

## Unit Testing

Unit testing is done to check whether the individual modules of the source code are working

properly. i.e. Testing each and every unit of the application separately by the developers in developers environment. Unit tests are created by programmers or occasionally by white box testers during the development process.

## User Acceptance Testing (UAT)

See "acceptance test" above.

## White Box Testing

Testing of software with complete knowledge of its internal code and logic. Contrast with "black box test" and "gray box test."

## Gamma Testing

Gamma Testing is done when the software is ready for release with specified requirements; this testing is done directly by skipping all the in-house testing activities.

## Question 2 – Explain Test bed and Test data

Test Bed is an execution environment which is configured for software testing. It consists of specific hardware, network topology, operating system, configuration of the product to be under test, system software and other applications. The test plan for a project should be developed from the test beds to be used.

Test Data is that run through a computer program to test the software. Test data can be used to test the compliance with effective controls in the software.

## Question 3 – Explain Severity and priority.

Severity is how seriously the bug is affecting the application. The severity type is defined by the tester based on the written test cases and functionality. Software Testing Question & Answer For Interview.

Priority is the order in which developer has to fix the bug. 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.

## Question 4 – Provide an example of a bug whose severity is high and priority is low

If an application or a 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 and priority is low.

## Question 5 - Provide an example of a bug whose severity is low and priority is high

If the company name is misspelled in the home page of a website, then the priority is high and the severity is low to fix it.

## Question 6 - When to stop testing?

- a) When all the requirements are adequately executed successfully through test cases
- b) Bug reporting rate reaches a particular limit
- c) The test environment no more exists for conducting testing
- d) The scheduled time for testing is over
- e) The budget allocation for testing is over

**Question 7 - Explain Desktop application testing, Client server application testing and Web application testing**

Desktop application runs on personal computers and work stations, so when you test the desktop application you are focusing on a specific environment. You will test complete application broadly in categories like GUI, functionality, Load, and backend i.e DB.

In client server application you have two different components to test. Application is loaded on server machine while the application exe on every client machine. You will test broadly in categories like, GUI on both sides, functionality, Load, client-server interaction, backend. This environment is mostly used in Intranet networks. You are aware of number of clients and servers and their locations in the test scenario.

Web application is a bit different and complex to test as tester don't have that much control over the application. Application is loaded on the server whose location may or may not be known and no exe is installed on the client machine, you have to test it on different web browsers. Web applications are supposed to be tested on different browsers and OS platforms so broadly Web application is tested mainly for browser compatibility and operating system compatibility, error handling, static pages, backend testing and load testing.

**Question 8 – Explain Performance testing with an example**

Performance testing is the testing, which is performed, to ascertain how the components of a system are performing, given a particular situation. Resource usage, scalability and reliability of the product are also validated under this testing Software Testing Question & Answer For Interview.

Example - We can test the application network performance on Connection Speed. A 70kb page would take not more than 15 seconds to load for a worst connection of 28.8kbps modem while the page of same size would appear within 5 seconds, for the average connection of 256kbps

**Question 9 – Explain Load testing with an example**

Load testing is meant to test the system by constantly and steadily increasing the load on the system till the time it reaches the threshold limit. It is the simplest form of testing which employs the use of automation tools such as Load Runner or any other good tools, which are available.

Example - To check the email functionality of an application, it could be flooded with 1000 users at a time. Now, 1000 users can fire the email transactions (read, send, delete,

forward, reply) in many different ways. If we take one transaction per user per hour, then it would be 1000 transactions per hour

## Question 10 – Explain Stress testing with an example

Stress testing is a type of performance testing focused on determining an application's robustness, availability, and reliability under extreme conditions. The goal of stress testing is to identify application issues that arise or become apparent only under extreme conditions. These conditions can include heavy loads, high concurrency, or limited computational resources.

Example - Excessive volume in terms of either users or data; examples a situation where a widely viewed news item prompts a large number of users to visit a Web site during a three-minute period

## Question 11 – What is entry and exist criteria in software testing?

Entry Criteria is the process that must be present when a system begins. Here are the Entry Criteria:-

- SRS
- FRS
- Test Case
- Test Plan

Exit Criteria ensure that the testing of the application is completed and ready for release. Here are

the exit criteria:-

- All the planned requirements must be met
- All the high Priority bugs should be closed
- All the test cases should be executed
- If the scheduled time out is arrived Software Testing Question & Answer For Interview.

MWP © 2015

- Test manager must sign off the release

## Question 12 – How do you know when to stop testing?

This can be difficult to determine when to stop testing. Many modern software applications are so complex and run in such an interdependent environment that complete testing can never be done. Common factors in deciding when to stop are:-

- Deadlines e.g. release deadlines, testing deadlines
- Test cases completed with certain percentage passed
- Test budget has been depleted
- Coverage of code, functionality, or requirements reaches a specified point
- Bug rate falls below a certain level
- Beta or alpha testing period ends

## Question 13 – What is Equivalence partitioning method?

In this method the input domain data is divided into different equivalence data classes. This method is typically used to reduce the total number of test cases to a finite set of testable test cases, still covering maximum requirements.

In short it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:

- 1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is going to be same. So one test case for valid input data should be sufficient.
- 2) Input data class with all values below lower limit. I.e. any value below 1, as a invalid input data test case.
- 3) Input data with any value greater than 1000 to represent third invalid input class.

Equivalence partitioning uses fewest test cases to cover maximum requirements

Question 14 – What is Boundary value analysis?

It's widely recognized that input values at the extreme ends of input domain cause more errors in system. More application errors occur at the boundaries of input domain.

'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis: Software Testing Question & Answer For Interview.

MWP © 2015

- 1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.
- 2) Test data with values just below the extreme edges of input domains i.e. values 0 and 999.
- 3) Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

Boundary value analysis is often called as a part of stress and negative testing.

Question 15 – What is compatibility testing?

Testing the application in different software and hardware environment is nothing but compatibility testing.

Question 16 – Is 'V' model better than 'Water Fall' model? If so how is it?

Yes 'V' model is better than 'Water Fall' model.

In Waterfall model only after coding the testing starts whereas in V model coding and testing happens simultaneously so it is Cost Effective.

Question 17 – What is 'Open' issue?

When you open an issue for the first filed it is a NEW. it is being tested by the project manager whether it is an issue or not. If it is an issue then the status changes to open.

Question 18- If a customer wants a new feature to be added, how would you go about adding that?

The following will be the process for the new features to be added:

1. the customer will prepare the detailed SRS document for the changes to be done.
2. This draft copy of document will be given to the 'Configuration Management' team.
3. Configuration management team will review the requirement for validity/feasibility.
4. The document is then given to the development team lead for effort estimation.
5. The effort estimation report (which covers the other effected functionalities (if any)) is provided to QA team.
6. The QA team then prepares and provides the effort estimation report for testing the new functionality completely.

7. This report is given to the configuration management team, which decides upon the cost estimation for the new functionality to implement.
8. This report is then given to the client for their review, acceptance & sign-off.
9. If customer agrees, the sign-off takes place.
10. After signoff development team starts coding the functionality and simultaneously QA team starts working upon writing test cases.

Software Testing Question & Answer For Interview.

MWP © 2015

Question 19 – What is the difference between Regression testing and Retesting?

Regression testing is the process of testing new bug fixes to ensure that they don't cause problems to occur involving problems fixed earlier. This process involves running a suite of tests.

Re-testing is the process of testing a single defect that was just fixed. Only one test is performed, and the goal is to make sure that the defect that was just fixed was, in fact, fixed properly.

Question 20 - What if the software is so buggy it can't really be tested at all?

The best bet in this situation is for the testers to go through the process of reporting whatever bugs or blocking-type problems initially show up, with the focus being on critical bugs. Since this type of problem can severely affect schedules, and indicates deeper problems in the software development process managers should be notified, and provided with some documentation as evidence of the problem.

Question 21 - What if there isn't enough time for thorough testing?

Use risk analysis to determine where testing should be focused. Since it's rarely possible to test every possible aspect of an application, every possible combination of events, every dependency, or everything that could go wrong, risk analysis is appropriate to most software development projects. This requires judgment skills, common sense, and experience. Considerations can include:

- Which functionality is most important to the project's intended purpose?
- Which functionality is most visible to the user?
- Which functionality has the largest safety impact?
- Which functionality has the largest financial impact on users?
- Which aspects of the application are most important to the customer?
- Which aspects of the application can be tested early in the development cycle?
- Which parts of the code are most complex, and thus most subject to errors?
- Which parts of the application were developed in rush or panic mode?
- Which aspects of similar/related previous projects caused problems?
- Which aspects of similar/related previous projects had large maintenance expenses?
- Which parts of the requirements and design are unclear or poorly thought out?
- What do the developers think are the highest-risk aspects of the application?
- What kinds of problems would cause the worst publicity?
- What kinds of problems would cause the most customer service complaints?
- What kinds of tests could easily cover multiple functionalities?
- Which tests will have the best high-risk-coverage to time-required ratio?

Question 22 – Explain about Security Testing

Security Testing



Security Testing tests the ability of the system/software to prevent unauthorized access to the resources and data. Security Testing needs to cover the six basic security concepts:

Confidentiality, integrity, authentication, authorization, availability and non-repudiation.

Software Testing Question & Answer For Interview.

MWP © 2015

**Confidentiality**

A security measure which protects against the disclosure of information to parties other than the intended recipient that is by no means the only way of ensuring the security.

**Integrity**

A measure intended to allow the receiver to determine that the information which it is providing is correct. Integrity schemes often use some of the same underlying technologies as confidentiality schemes, but they usually involve adding additional information to a communication to form the basis of an algorithmic check rather than the encoding all of the communication.

**Authentication**

The process of establishing the identity of the user. Authentication can take many forms including but not limited to: passwords, biometrics, radio frequency identification, etc.

**Authorization**

The process of determining that a requester is allowed to receive a service or perform an operation. Access control is an example of authorization.

**Availability**

Assuring information and communications services will be ready for use when expected. Information must be kept available to authorized persons when they need it.

**Non-repudiation**

A measure intended to prevent the later denial that an action happened, or a communication that took place etc. In communication terms this often involves the interchange of authentication information combined with some form of provable time stamp.

## Question 23 - What is Exploratory Testing?

In exploratory testing tester focuses more on how the software actually works, testers do minimum planning and maximum execution of the software by which they get in depth idea about the software functionality, once the tester starts getting insight into the software he can make decisions to what to test next. Exploratory testing is mostly used if the requirements are incomplete and time to release the software is less.

## Question 24 – What is Maintenance Testing?

Maintenance Testing is done on the already deployed software. The deployed software needs to be enhanced, changed or migrated to other hardware. The Testing done during this enhancement, change and migration cycle is known as maintenance testing. Once the software is deployed in operational environment it needs some maintenance from time to time in order to avoid system breakdown, most of the banking software systems needs to be operational 24\*7\*365. So it is very necessary to do maintenance testing of software applications. In maintenance testing, tester should consider 2 parts:-

- 1) Any changes made in software should be tested thoroughly.
- 2) The changes made in software do not affect the existing functionality of the software, so regression testing is also done.

Software Testing Question & Answer For Interview.

MWP © 2015

Question 25 - Why is Maintenance Testing required?

User may need some more new features in the existing software which requires modifications to be done in the existing software and these modifications need to be tested. End user might want to migrate the software to other latest hardware platform or change the environment like OS version, Database version etc. which requires testing the whole application on new platforms and environment.

Question 26 – What is SSL?

SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. SSL is an industry standard and is used by millions of websites in the protection of their online transactions with their customers.

Typically an SSL Certificate will contain your domain name, your company name, your address, your city, your state and your country. It will also contain the expiration date of the Certificate and details of the Certification Authority responsible for the issuance of the Certificate. When a browser connects to a secure site it will retrieve the site's SSL Certificate and check that it has not expired, it has been issued by a Certification Authority the browser trusts, and that it is being used by the website for which it has been issued. If it fails on any one of these checks the browser will display a warning to the end user letting them know that the site is not secured by SSL.

Question 27 – What is Concurrency Testing?

Concurrency Testing commonly known as Multi User Testing is used to know the effects of accessing the Application, Code Module or Database by different users at the same time.

Question 28 – What is the difference between High level and Low level test cases?

High level Test cases are those which cover major functionality in the application (i.e. retrieve, update display, cancel and database test cases)

Low level test cases are those related to User Interface (UI) in the application.

## Question 29 – What is ‘AUT’?

AUT is nothing but ‘Application Under Test’. After the designing and coding phase in software development life cycle, the application comes for testing then at that time the application is stated as Application Under Test.

## Question 30 – What is Bug Life Cycle?

Bug life cycle is nothing but the various phases a bug under goes after it is raised or reported.

The different phases of Bug life cycle are,

- New or Opened
- Assigned
- Fixed
- Tested
- Closed

## Question 30 - What is the difference between Bug, Error and Defect?

**Bug** – It is found in the development environment before the product is shipped to the respective customer.

**Error** – It is the Deviation from actual and the expected value.

**Defect** – It is found in the product itself after it is shipped to the respective customer

## Question 31 – What is Negative testing?

Testing the system using negative data is called negative testing. For e.g. testing the password where it should minimum of 8 characters so testing it using 6 characters is negative testing.

## Question 32 – What are cookies?

It is a small piece of information that server keeps at the browser side so that when request is made to the server next time then previous settings can be remembered for e.g. most of the time the username and password are kept in cookies so that when user tries to open the page then the field displays the username and password.

## Question 33 – What is Test Server?

The place where the developers put their development modules, which are accessed by the testers to test the functionality.

Question 34 - What difference you have observed while testing the Client/Server application and web

Client server application is an intranet application having two tires. Here client server and database will be there. Web server application is an internet application having 3tire architecture. Here client server, web server and database will be there.

Testing is same for both client server and web based application but for web based application

we have to test in various browsers like IE 7, IE 8, Mozilla, and Netscape.

Question 35 - If a bug is found that is not replicable all times, does that bug should be logged in the bug tracker tool or does it reported to developer?

We don't log a defect. First of all we need to investigate the issue and should clarify that is it a defect or not. If we cannot be able to investigate the issue or the issue is beyond our limits then we need to send this issue to our TL/Manager. A TL/Manager has to identify that this issue is really a defect or user mistake. If they says that it is a defect then and then only we need to log it as a defect.

Question 36 – Throughout your software testing / quality assurance career which was the most critical defect you have logged. Please give some brief idea about the defect logged with the solution provided by the development team.

Question 37 – What is defect leakage?

Defect leakage means defect found/reproduced by client/user which the tester was unable to re-produce.

Question 38 - What are the different types of tests you perform in system testing?

In system testing we can perform Functionality, Regression and Performance.

Question 39 – What is the difference between Monkey testing and Ad hoc testing?

Monkey testing is the random testing. Here we don't know about the application.

Ad-hoc is informal testing where we know about the application well in hand.