MANUAL TESTING

MANUAL TESTING

TOPIC - 01

FUNDAMENTALS OF TESTING

MANUAL TESTING

01. Introduction

- ❖ How projects are taken in the industry : Ex : Infosys & ICICI Bank
- ❖ Software Development Life Cycle

02. Necessity of Testing

- ❖ Software Systems are now part of our everyday life. They are used almost everywhere, for example in:
    - Banking & Financial Institute
    - Retail Industry
    - Central & Local Government
    - Transport (Planes, Trains & Automobile)
    - Medicines (Hospital, Research Centre)
- ❖ Software System Failures can lead to:
    - **Human Injury or Death**
      E.g. airplanes crashing
    - **Technological disasters**
      E.g. Missile Systems malfunctioning
    - **Legal action and associated costs**
      E.g. failure to meet contractual obligations
    - **Loss of face for suppliers and/or their customers**
      E.g. mis-spelling company name on mail shots
- ❖ 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

03. History of Testing

- ❖ In 1993, **Disney world** launched a CD which had many interactive games. All the CDs were sold on the very same day. The next day itself, lot of complaints were received from end users. End users were not able to run the CD.
- ❖ Executive Management were surprised by this statement and started to analyze the cause of the problem. In their root cause analysis they found that the problem was due to **Inadequate Testing**.
- ❖ The software was testing only on **Windows O.S.** of Microsoft and many end users were not able to run the CD because they had **Macintosh O.S.** of Apple.
- ❖ As the company did not conduct **Compatibility Testing** with all the O.S. available. There was huge loss of money touching billion's of dollars and more importantly huge loss of Reputation.

04. Bug is always there...

- A good conversation between the person who is **Master** in testing and the person who is **New in Testing** (**Novice**) :

- The master: "Any program, no matter how small contains bugs".
- The novice did not believe the master's words.
- "What if the program were so small that it performs a single function?" he asked.

- "Such a program would have no meaning, but if such a one existed the operating system would fail eventually, producing a bug." But the novice was not satisfied.
- "What if the operating system did not fail" he asked.
- "There is no operating system that does not fail", said the master, "but if such a one existed, the hardware would fail eventually producing a bug". The novice was still not satisfied.

- "What if the hardware did not fail?" he asked.
- "There is no hardware that does not fail, he said, but if such a one existed, then the user would want the program to do something different and this too is a bug".

- "A program without bugs would be an absurdity, a no one such If there were a program without any bug, then the world would cease to exist".

- **Hence, the bug always exists in the system and we can not deliver 100% defect free system to the customer.**

05. What is testing ? Why testing ? Testing How ?

- Testing is the process of executing a program with the intent of finding errors.
- To find greatest possible number of errors with manageable amount of efforts applied over a realistic time span with a finite number of test cases.
- "Testing is the **destructive** activity for some new **construction** activity."

- Contribute to the delivery of higher quality software product
- Undetected errors are costly to detect at a later stage
- Satisfied users and to lower maintenance cost

- By examining the users' requirements
- By reviewing the artifacts like design documents
- By examining the design objectives
- By examining the functionality
- By examining the internal structures and design
- By executing code

06. Principles of Testing

- **Economics of Testing -** It is both the driving force and the limiting factor

- **Driving** – Testing should be planned and started at the early phases of SDLC.

- **Limiting** - Testing must end when the economic returns cease to make it worth while i.e. the costs of testing process significantly outweigh the returns.

- **Pareto Principle** - is applicable for testing which states that 80% of the errors are found in 20% of the program component.

- **Avoid Testing** - A programmer should avoid Testing of his/her own program.

07. Limitations of Software Testing

- Even if we could generate the input, run the tests, and evaluate the output, we would not detect all faults.
- Correctness is not checked.
- The programmer may have misinterpreted the specs; the specs may have misinterpreted the requirements.
- There is no way to find missing paths due to coding errors.

08. Psychology of Testing & Quality of a good Tester

- Test Engineers pursue defects not people
- Don’t assume that no error(s) will be found
- Test for Valid and Expected as well as Invalid and Unexpected
- The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section
- Testing is extremely creative and intellectually challenging

- **Good Testers are :**
  - ✓ Explorers
  - ✓ Creative
  - ✓ Perfectionists
  - ✓ tactful
  - ✓ Diplomatic
  - ✓ They exercise good judgments
  - ✓ The best tester isn’t one who finds the most bugs or who embarrasses the most developers, but who gets the most bugs fixed.

09. Test Case / Test Suite / Test Cycle

- **Test Case :**
  - ✓ "A set of **test inputs, execution conditions,** and **expected results** developed for a particular objective, such as to exercise a particular program path or to verify compliance with a **specific requirement.**" (…IEEE)
  - ✓ In other words, a planned sequence of actions (with the objective of finding errors)

- **Test Suite :**
  - ✓ A set of individual test cases/scenarios that are executed as a package, in a particular sequence and to test a particular aspect.
  - ✓ E.g. Test Suite for a GUI or Test Suite for functionality

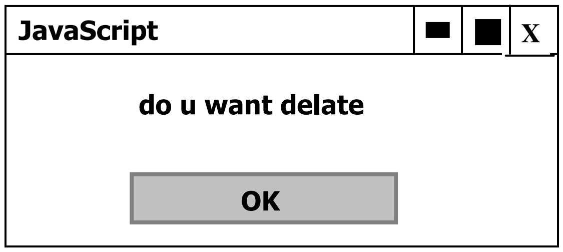
- **Test Cycle :**
  - ✓ A test cycle consists of a series of test suites which comprises a complete execution set from the initial setup to the the test environment through reporting and clean up.
  - ✓ E.g. Integration test cycle / regression test cycle

10. Testing Process

- Establish the Test Objectives
- Develop test plan
- Design Test Cases
- Review Test Cases
- Execute the tests
- Examine the test results
- Defect Logging
- Preparing Test Reports

11. Scenario for Testing

- Please note done the defect in the message box (Testing a message box)



TOPIC – 02

QUALITY

01. Definitions

02. Quality Attributes

03. Quality in Perception

04. Quality in Fact

05. Quality Culture

06. Quality Management System – QA & QC

07. Quality Assurance (QA)

08. Quality Control (QC)

09. Difference between QA & QC

10. Professional Standards

- ISO - International Organization for Standardization
- IEEE - Institute of Electrical & Electronics Engineers
- ANSI - American National Standards Institutes
- CMMi - Capability Maturity Model Integration
- Other Standards

MANUAL TESTING

01. Definitions

- **Customer Point of View**: Software should be fit for use.

- **Developer Point of View:** Software should meet customer requirement.

02. Quality Attributes

- Correctness
- Efficiency
- Integrity
- Maintainability
- Reliability
- Usability

03. Quality in Perception

- Treating every customer with integrity, Courtesy and respect
- Meeting the customer's expectations
- Satisfying customer's needs
- Delivering the right product

04. Quality in Fact

- Doing the right thing
- Doing it the right way
- Doing it the right the first time
- Doing it on time

05. Quality Culture

- Listening to customers to determine their requirement
- Identifying costs of quality and focusing on prevention
- Doing the right thing right the first time
- Continuing process improvement
- Taking ownership at all levels of the organization
- Demonstrating executive leadership and commitment

06. Quality Management System (QMS) – QA & QC

- QMS is the process of preventing defects in the software process. To ensure that there are no defects in the final product, the whole quality process is divided into two parts.
  (01) Quality Assurance
  (02) Quality Control

07. Quality Assurance (QA)

- QA is based on process where we measure each processes, identify any weakness and suggest improvement.
- The out put of QC activities often becomes the input to QA.

08. Quality Control (QC)

- QC is the activity based on product where we measure the product, identify any weakness and improve the product.
- The change may include from single line code to reworking the whole product.

09. Difference between QA & QC

Quality Assurance (QA)	Quality Control (QC)
Process Oriented	Product Oriented
Preventive Approach	Detective Approach
E.g. Audit, Review	E.g. Testing

10. Professional Standards

ISO - International Organization for Standardization

- ✓ ISO is based in Switzerland.
- ✓ ISO is the world's largest developer of standards.

- ✓ **ISO 9001 : 2000**
  Applies to many kinds of production & Manufacturing organization, not just Software. It covers documentation, design, development, production, testing, Installation, servicing and other processes. The full set of stds consists of :

- ✓ Q9001-2000 – Quality Management System – Requirements
- ✓ Q9000-2000 – Quality Management System – Fundamentals & Vocabulary
- ✓ Q9004-2000 – Quality Management System – Guidelines for Performance Improvement.

- ✓ To be certified, a third party auditor assesses and organization, and certification is typically good for about 3 years, after which a complete reassessment is required. Note that ISO certification does not necessarily indicate quality product- it indicates only that documented processes are followed.

❖ IEEE - Institute of Electrical & Electronics Engineers

- ✓ IEEE publishes a body of standards that address software engineering and maintenance. A portion of these standards addresses software testing and test reporting.
- ✓ IEEE among other things, create standards such as :
  (01) IEEE standard for software test documentation – IEEE/ANSI Standard 829
  (02) IEEE standard of software unit testing – IEEE/ANSI Standard 1008
  (03) IEEE std for software quality assurance plans – IEEE/ANSI Standard 730

❖ ANSI - American National Standards Institutes

- ✓ ANSI is the primary industrial standards body in the U.S. publishes some software-related standards in conjunction with the IEEE and ASQ (American Society for Quality)

❖ CMMi - Capability Maturity Model Integration

- ✓ CMMi is certification provided to Software Development Company depending on their maturity to develop quality software developed by the software engineering institute (SEI).
- ✓ This model can be used to assess an organization against a scale of five process maturity levels. Each level ranks the organization according to its standardization of processes in the subject area being assessed.
- ✓ There are five levels of the CMMi according to the SEI.

(01) Level – 1 : Initial Level (Ad-hoc)

- ✓ Characterized by chaos, periodic panics and heroic
- ✓ Efforts required by individuals to successfully complete projects.
- ✓ Few if any processes in place
- ✓ Success may not be repeatable.

(02) Level – 2 : Repeatable

- ✓ Software project tracking, Requirement Mgt, Realistic Planning and Configuration Mgt processes are in place.
- ✓ Successful practices can be repeatable.

(03) Level – 3 : Defined

- ✓ Standard Software Development and maintenance processes are integrated throughout an organization
- ✓ A software engineering process group is in place to oversee software processes
- ✓ Training programs are used to ensure the understanding and compliance.

(04) Level – 4 : Managed (Measured)

- ✓ Metrics are used to track productivity, processes & Products
- ✓ Project performance is predictable
- ✓ Quality is consistently high

(05) Level – 5 : Optimizing

- ✓ Focus is on continuous process improvement
- ✓ The impact of new processes and technologies can be predicated and effectively implemented when required.

❖ Other Standards

- ✓ SPICE
- ✓ Trillium
- ✓ TickIT
- ✓ BootStrap
- ✓ ITIL
- ✓ MOF
- ✓ CobiT

====**Topic - 2**=== XXX ==========**Q U A L I T Y ==**===== XXX ============

TOPIC – 03

Software Development Life Cycle

01. SDLC

02. Product Based SDLC

03. Project Based SDLC

04. Phases in SDLC

MANUAL TESTING

01. SDLC (Software Development Life Cycle)

- The software Development Life Cycle (SDLC) is a flow that demonstrates different stages in building quality software products. It states the staring point and the ending point of any software development.
- Each phase in the development process has a defined input and output. The software developed by different Software companies can be broadly categorized into two types :
  (01) Product Based Software Development
  (02) Project Based Software Development

02. Product Based SDLC

- The software company does a feasible study and then develops the software product for particular industry with the standard requirement and sells this product to all the companies that belong to the same industry.
- E.g. Accounting Software – "**Tally**"
- E.g. Banking Software – "**Flex Cube**". Iflex is CMMi Level 5 Software Development Company that has developed a Banking software called "Flex Cube" for Banking Industry which is widely used in US and UK

03. Project Based SDLC

- The software companies develop software as per the requirement specification provided by the customer. There is contractual agreement between Software Company and the Customer, after the software has been development, the software company handovers the software to the customer.
- E.g. Infosys, TCS, Wipro, Patni etc develops software as per the requirement provided by the customers such as G.E, Citibank etc..

04. Phases in SDLC

- SDLC has 5 phases. Each phases has its input that needs to be processed in order to receive desired output.
- Phases are :
  (01) Requirement Analysis
  (02) Software Design
  (03) Coding
  (04) Testing
  (05) Maintenance

❖ **Requirement Analysis :**

- ✓ In this phase, the requirement of the clients are obtained, analyzed, documented and validated.
- ✓ Requirement should be clear and precise to produce high quality software products.
- ✓ 95% of the software fails due to ambiguous requirement and incomplete SRS (Software Requirement Specification Document)
- ✓ Some of the mistakes commonly are :
  - (01) Start Development work without understanding the requirement correctly.
  - (02) Whenever there are any changes made to the requirement, are not incorporated to SRS document.
  - (03) Requirements are accepted without understanding it's impact on the software

- ✓ **Input :-** Customer Requirement
- ✓ **Process :-** Analysis of the Requirement
- ✓ **Output :-** Software Requirement Specification (SRS) document

❖ **Software Design :**

- ✓ Software design involves development of architecture, flow of the application, logic behind each module and data structure.
- ✓ It also involves detail study of hardware and the software.
- ✓ While designing the software, it needs to consider different aspect of an application such as usability, performance, security, compatibility and the database.
- ✓ Unified Modeling Language (UML) has become standard way of documenting component of Software Design.

- ✓ **Input :-** Software Requirement Specification (SRS) document
- ✓ **Process :-** Designing the Software
- ✓ **Output :-** (01) Software Design Document (SDD)Requirement
  (02) Internal Design Document (IDD)
  OR
  (03) Detailed Design Document (DDD)

❖ **Coding :**

- ✓ After the software Design, the developer develops the code as per the design specification.
- ✓ Desktop Applications are developed in programming languages such as C, C++, Java, .Net etc..
- ✓ Web Applications are developed in scripting languages like ASP.Net, PHP, CGI, Perl etc..

- ✓ **Input :-** (01) Software Design Document (SDD)Requirement
  (02) Internal Design Document (IDD)
  OR
  (03) Detailed Design Document (DDD)
- ✓ **Process :-** Developing Code
- ✓ **Output :-** Source Code

❖ **Testing :**

- ✓ In this phase, the software tested to ensure that it meets customer requirement.
- ✓ Initially each unit or module is tested independently and then it is integrated to perform Integration Testing, then follows System Testing and User Acceptance Testing (UAT)

- ✓ **Input :-** Executable Code (Program) & Test condition
- ✓ **Process :**- Testing
- ✓ **Output :**- Test Cases & Defect Log

❖ **Maintenance :**

- ✓ In this phase, the customer has started using the software and need some enhancement or modification in the existing system.
- ✓ Again the requirements to be analyzed, design, develop the code & test the changed code as well the previous code. i.e. Regression Testing.

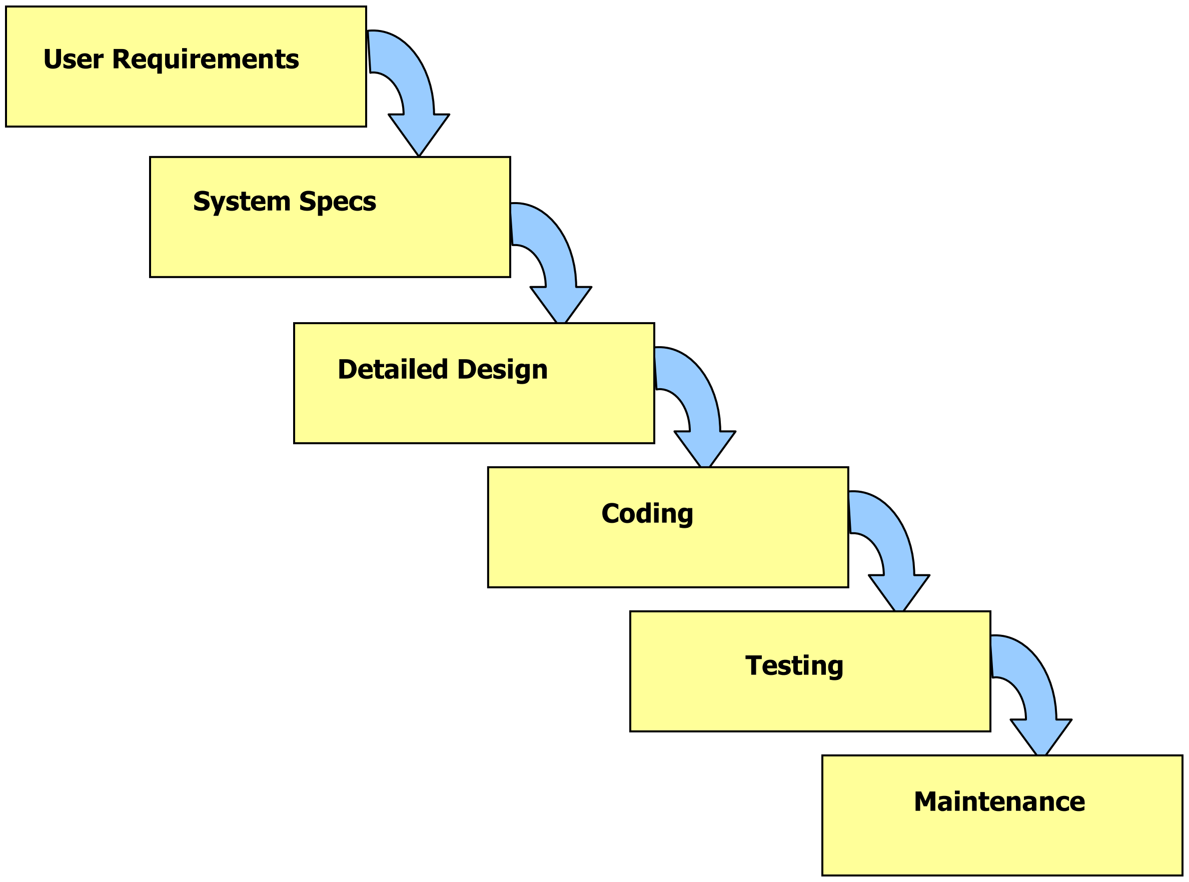
- ✓ **Input :-** Version 1.0
- ✓ **Process :**- Requirement Analysis, Software Design, Implementation/Coding, & Testing
- ✓ **Output :**- Version 2.0

====**Topic - 3**========= XXX =====**S D L C**======== XXX ============

TOPIC – 04

Software Development Models

01. Waterfall Model



- The name "**Waterfall**" portrays system progress flows from the top to the bottom, like water falling down the steps in a waterfall panorama, one phase at a time towards the bottom in a cascading effect.
- This model is also called as "**Linear Sequential Model**"
- In this model, the requirements are freeze and detailed SRS document is prepared to carry software design, implementation or coding, testing and then maintenance.
- Some unique features are :
  (01) It leads to a concrete and clear approach to software development.
  (02) Documentation is produced at every stage of model which is very helpful for people who are involved.
  (03) The steps are discrete; there is no overlap.
  (04) The waterfall model is the oldest and the most widely used paradigm.
- The waterfall model derives its name due to the cascading effect from one phase to the other as is illustrated in the figure. In this model, each phase well defined starting and ending point, with identifiable deliveries to the next phase.

- The model consist of 6 distinct phases :
  (01) Requirement Analysis Phase
  (02) System Specifications Phase
  (03) Detailed Design Phase
  (04) Coding Phase
  (05) Testing Phase
  (06) Maintenance Phase

- **In the Requirement Analysis Phase**

  - ✓ The problem is specified along with the desired service objectives (Goals)
  - ✓ The constraints are identified.

- **In the System Specifications Phase**

  - ✓ The system specification is produced from the detailed definition of Goals and constraints of the Requirement phase
  - ✓ Note that in some text, the requirements analysis and specifications phases are combined and represented as a single phase.

- **In the Detailed Design Phase**

  - ✓ The system specifications are translated into a software representation.
  - ✓ The software engineer at this stage is concerned with :
    (01) Data Structure
    (02) Software Architecture
    (03) Algorithmic Detail
    (04) Interface representations
  - ✓ The hardware Req. are also determined at this stage along with a picture of the overall system architecture.
  - ✓ By the end of this stage, the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces.
  - ✓ Any faults in the specification should ideally not be passed "down stream".

- **In the Coding Phase**

  - ✓ The designs are translated into the software domain.
  - ✓ Detailed documentation from the design phase can significantly reduce the coding effort. Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.

- **In the Testing Phase**

  - ✓ All the program units are integrated and tested to ensure that the complete system meets the software requirements.
  - ✓ After this phase, the software is delivered the customer.

- **Maintenance Phase**

  - ✓ This phase is usually longest phase of the software.
  - ✓ In this phase, the software is updated to :
    (01) Meet the changing customer needs
    (02) Adapted to accommodate changes in the external environment
    (03) Correct errors and oversights previously undetected in the testing phases
    (04) Enhancing the efficiency of the software

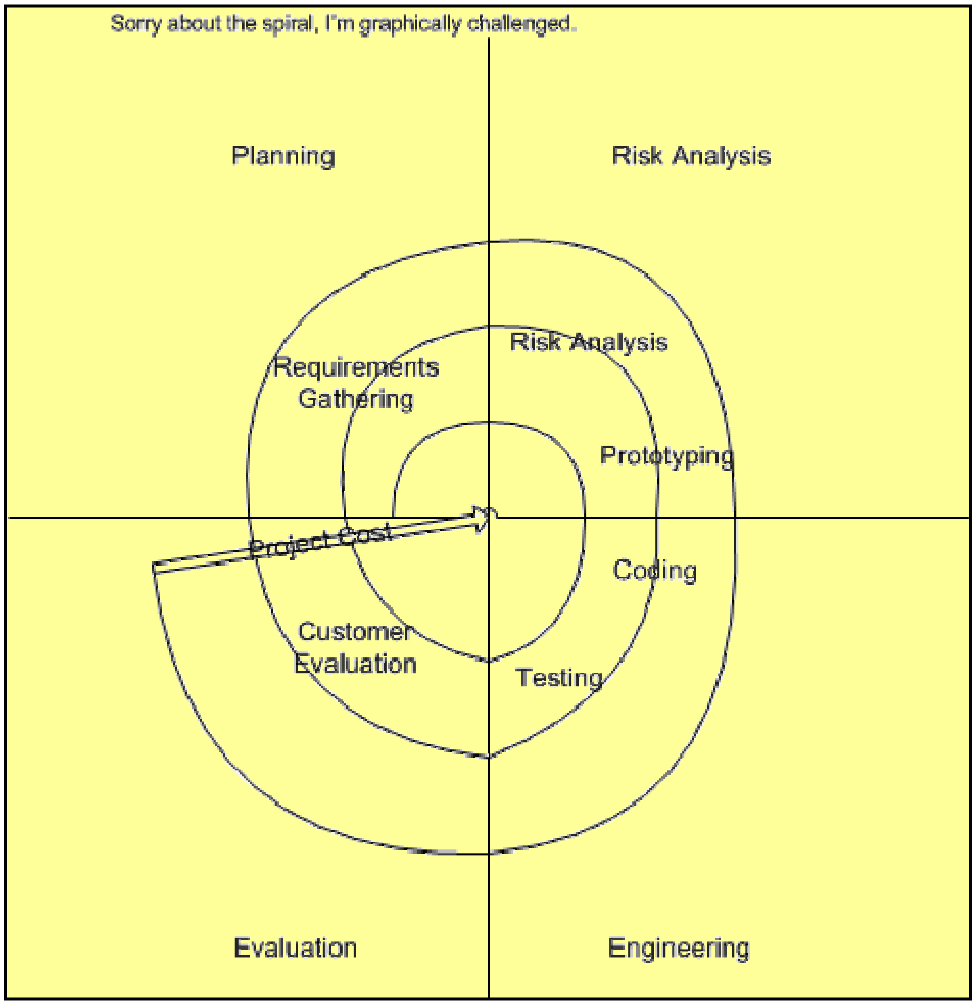
- **Advantages**

  - ✓ It is an enforced disciplined approach
  - ✓ It is documentation driven, that is, documentation is produced at every stage
  - ✓ Simple & easy to use.

- **Disadvantages**

  - ✓ Rigid Model for Use.

02. Spiral Model



The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase.

Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

Advantages

- ✓ High amount of risk analysis
- ✓ Good for large and mission-critical projects.
- ✓ Software is produced early in the software life cycle.

Disadvantages

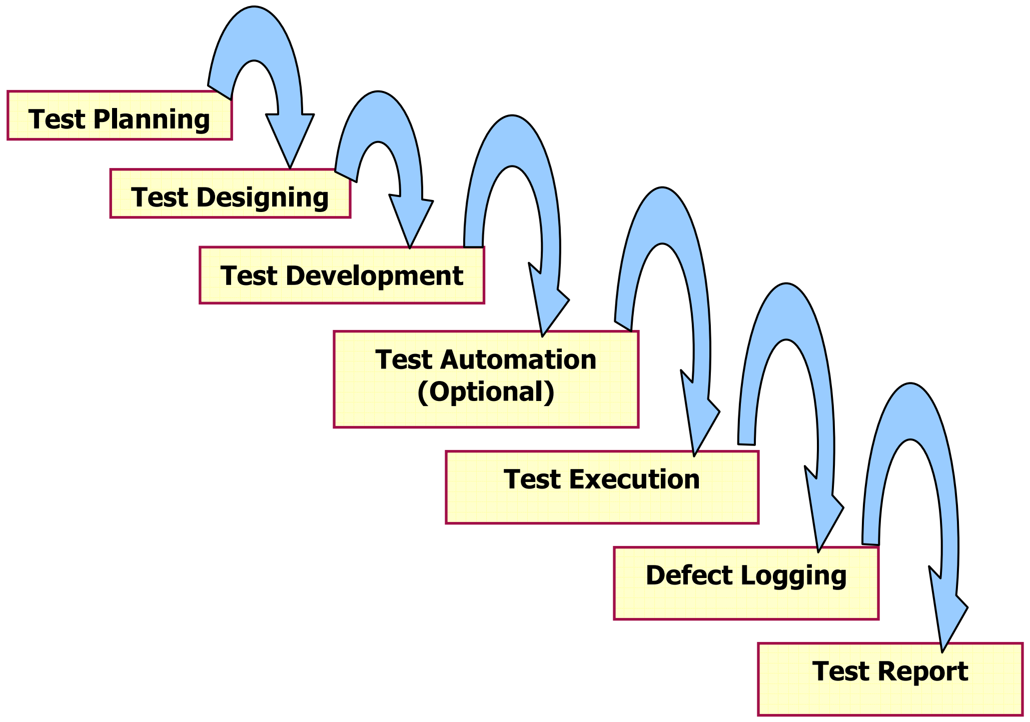
- ✓ Can be a costly model to use.
- ✓ Risk analysis requires highly specific expertise.
- ✓ Project's success is highly dependent on the risk analysis phase.
- ✓ Doesn't work well for smaller projects.

And that's it. If you have any input, especially your views on advantages and disadvantages of any particular model, feel free to leave them in the comments and I can add them to my copy.

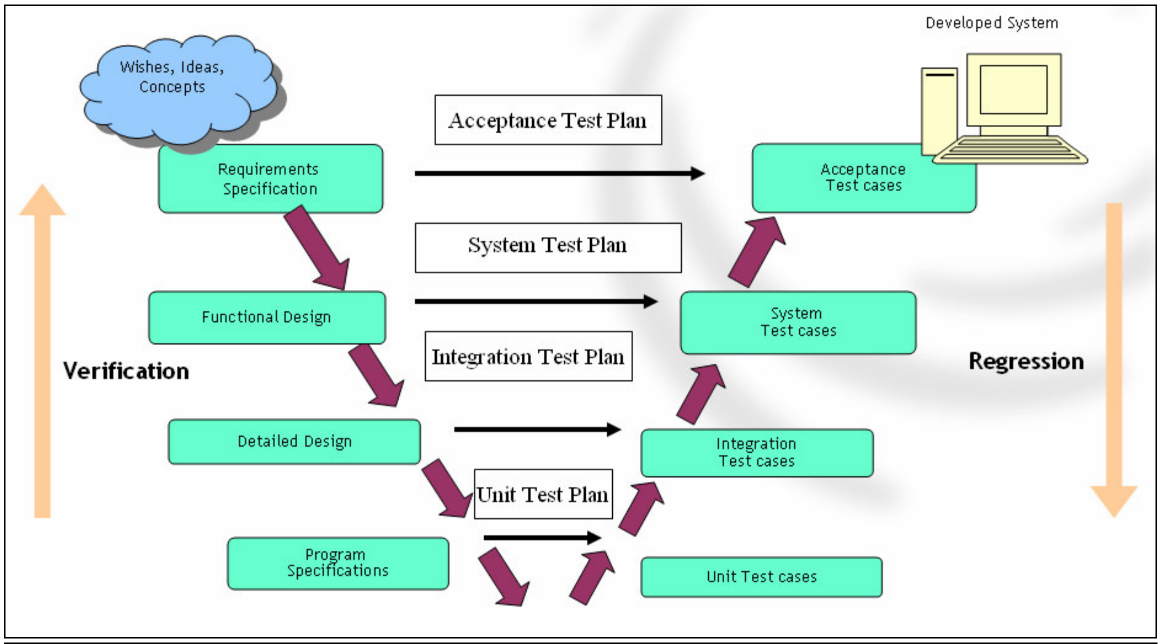
03. Prototype Model

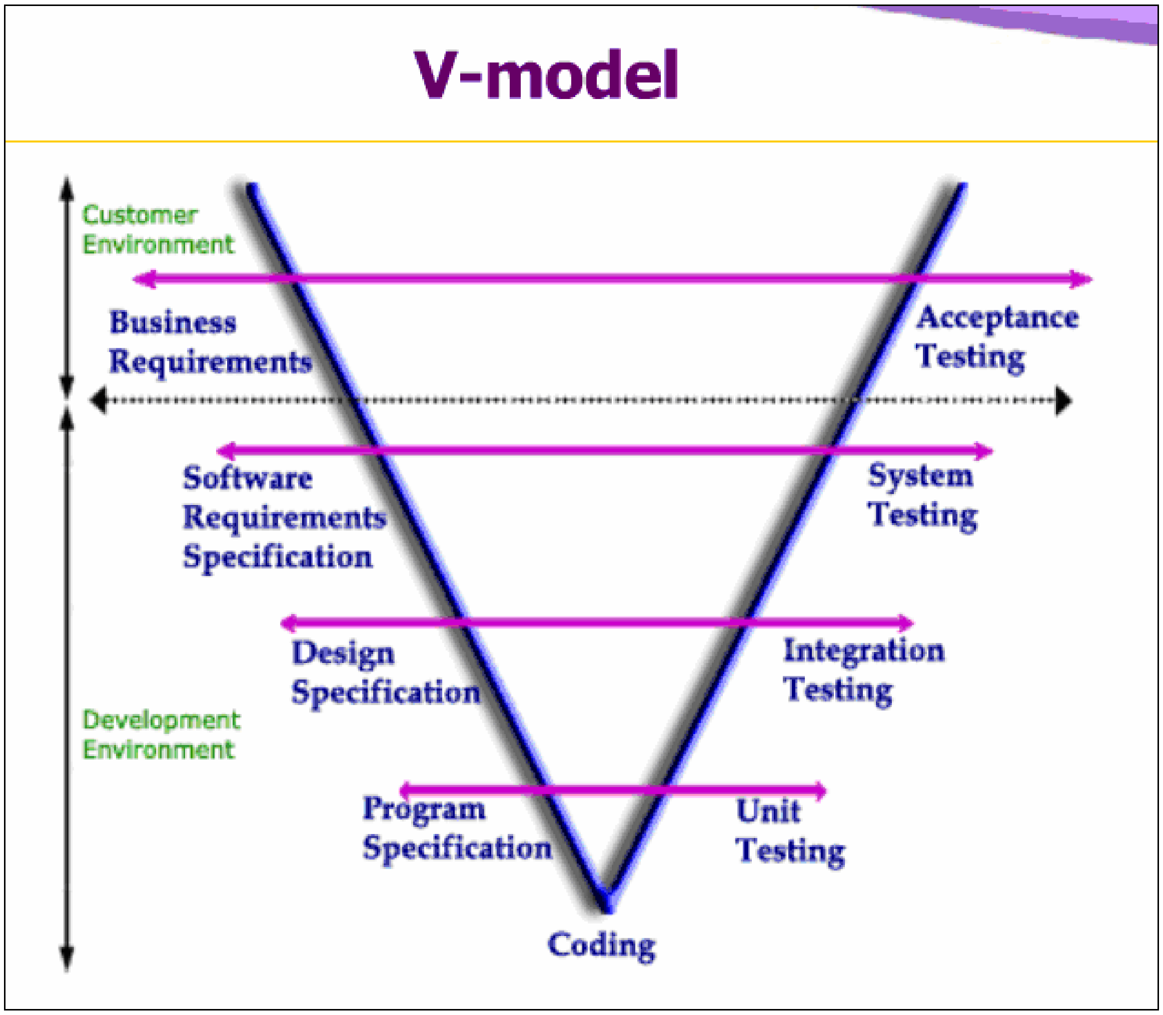
- ✓ For some object, it becomes difficult to obtain exact requirement.
- ✓ In this case, we build a demo or prototype and demonstrate to the customer.
- ✓ Based on the user feedback, application is build and SRS document is prepared.
- ✓ The idea behind this model is to finalize SRS document.
- ✓ It is to prove your design concepts and capability in terms of delivering the quality.
- ✓ Prototype is generally developed at the cost of the software company, so it is done with minimum resources.

04. Software Testing Life Cycle (STLC)



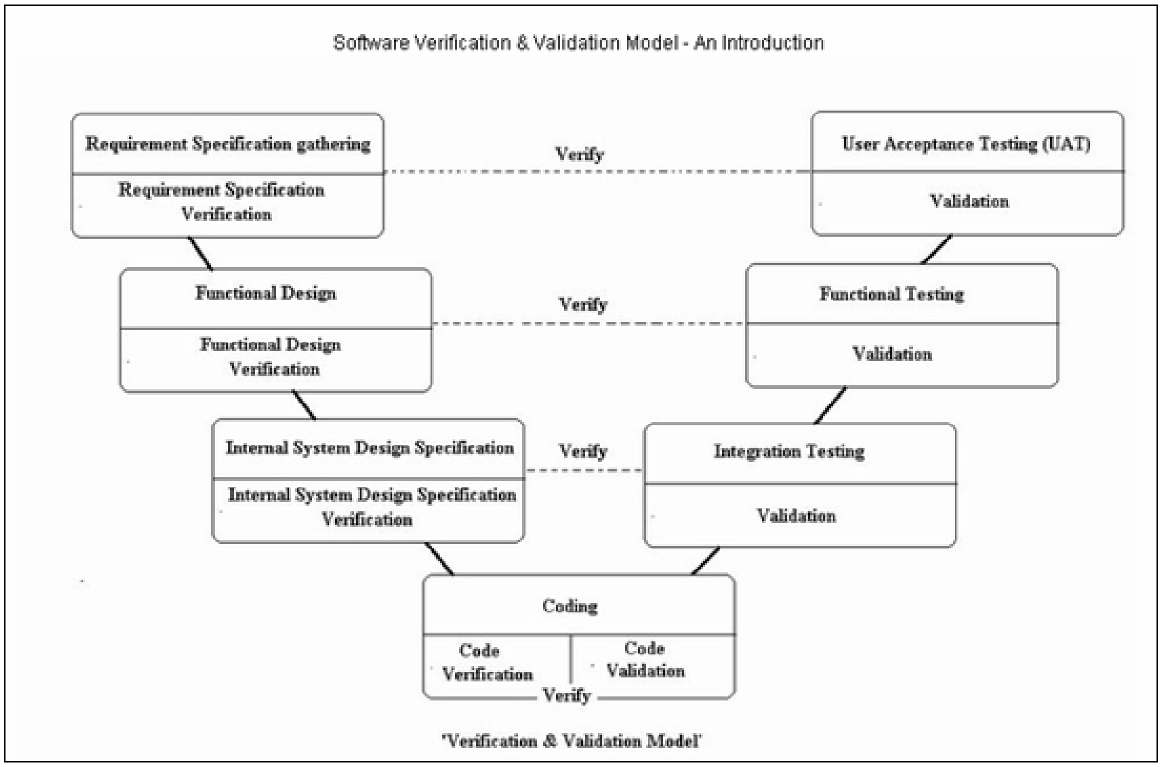
05. V- Model (Validation - Model)

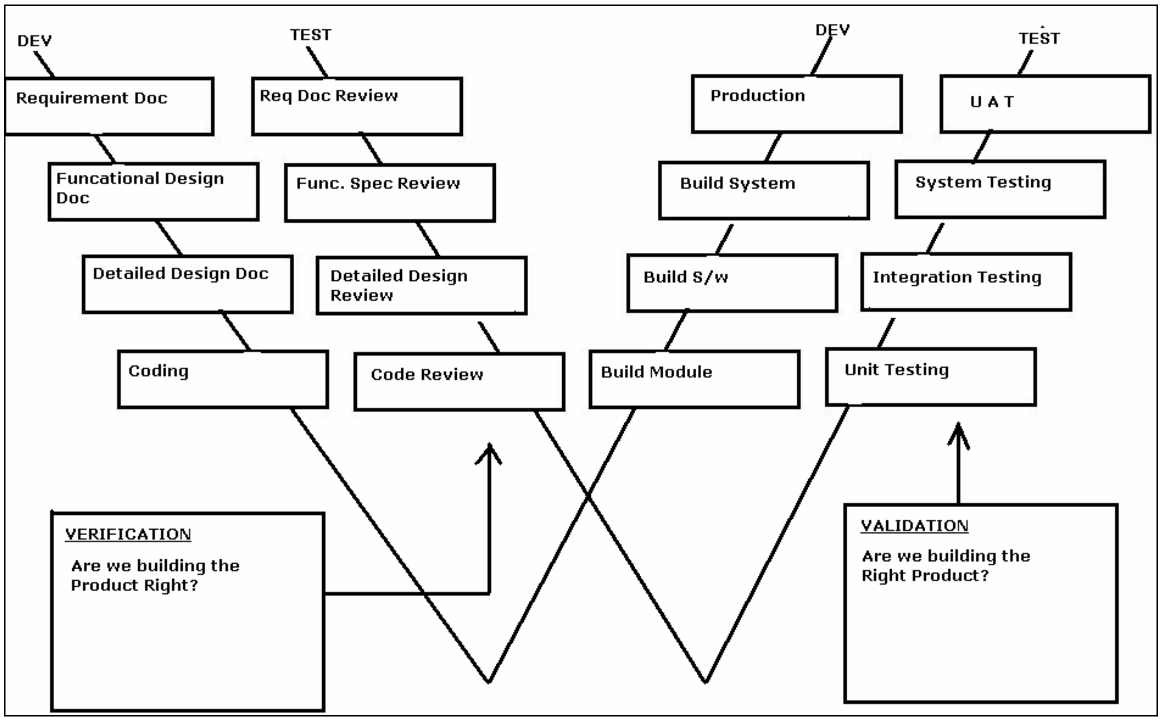




- There are some distinct test phases that take place in each of the software life cycle activity
- It is easier to visualize through the famous **Waterfall** model of development and **V-model** of testing
- The V proceeds from left to right, depicting the basic sequence of development and testing activities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

06. V & V – Model (Verification & Validation – Model)


Software Verification & Validation Model - An Introduction
Requirement Specification gathering
Requirement Specification Verification
Verify
User Acceptance Testing (UAT)
Validation
Functional Design
Functional Design Verification
Verify
Functional Testing
Validation
Internal System Design Specification
Internal System Design Specification Verification
Verify
Integration Testing
Validation
Coding
Code Verification
Code Validation
Verify
'Verification & Validation Model'



- **VERIFICATION (FDA Definition)**
  Verification is defined in FDA's Quality System Regulation (21 CFR 820.3) as "confirmation by examination and provision of objective evidence that specified requirements have been fulfilled." In a software development environment, software verification is confirmation that the output of a particular phase of development meets all of the input requirements for that phase.
- **VALIDATION (FDA Definition)**
  Software validation is "establishing by objective evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements" [Ref: NIST 500-234]. Software validation is essentially a design verification function as defined in FDA's Quality System Regulation (21 CFR 820.3 and 820.30) and includes all of the verification and testing activities conducted throughout the software life cycle.

  Design validation encompasses software validation, but goes further to check for proper operation of the software in its intended use environment.

- **VERIFICATION (IEEE Definition)**
  "Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled."

- **VALIDATION (IEEE Definition)**
  "Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled."

- **Using the above definitions in software development**
  Validation, in its simplest terms, is the demonstration that the software implements each of the software requirements correctly and completely. In other words, the "right product was built." Verification is the activity which ensures the work products of a given phase fully implement the inputs to that phase, or "the product was built right."

- **Levels of Verification**
  There are four levels of verification:

  - ✓ **Inspection**
    Typical techniques include desk checking, walkthroughs, software reviews, technical reviews, and formal inspections (e.g., Fagan approach).
  - ✓ **Analysis**
    Mathematical verification of the test item, which can include estimation of execution times and estimation of system resources.
  - ✓ **Testing**
    Also known as "white box" or "logic driven" testing. Given input values are traced through the test item to assure that they generate the expected output values, with the expected intermediate values along the way. Typical techniques include statement coverage, condition coverage, and decision coverage.
  - ✓ **Demonstration**
    Also known as "black box" or "input/output driven" testing. Given input values are entered and the resulting output values are compared against the expected output values. Typical techniques include error guessing, boundary-value analysis, and equivalence partitioning.

- **Levels of Validation**

  - ✓ **Unit Testing**
    Testing conducted to verify the implementation of the design for one software element (unit, module) or a collection of software elements.
  - ✓ **Integration Testing**
    An orderly progression of testing in which various software elements and/or hardware elements are integrated together and tested. This testing proceeds until the entire system has been integrated.
  - ✓ **System Testing**
    The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.
  - ✓ **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.

- **Using the above definitions in software development**
  Validation, in its simplest terms, is the demonstration that the software implements each of the software requirements correctly and completely. In other words, the "right product was built." Verification is the activity which ensures the work products of a given phase fully implement the inputs to that phase, or "the product was built right."

- **Explanation**
  The four methods for verification can be used at any of the levels although some work better than others for a given level of verification. As an example, the most effective way to find anomalies at the component level is inspection. On the other hand, inspection is not applicable at the system level (you don't look at the details of code when performing system level testing). A logical approach to testing is to utilize techniques and methods that are most effective at a given level.

  Component level verification can easily get very expensive. Companies need to avoid making statements like "all paths and branches will be executed during component testing." These statements make for a very expensive test program, as all code developed is required to have one of the most labor-intensive type of testing performed on it. To minimize the costs of component verification, the V&V group develops rules for determining the type of verification method(s) needed for each of the software functions. As an example, very low complexity software function, which is not on the safety critical list, may only need informal inspection (walkthrough) performed. Other complicated functions typically require white box testing since the functions become difficult to determine how they work. We recommend performing inspections before doing the white box testing for a given module as it is less expensive to find the errors earlier in the development.

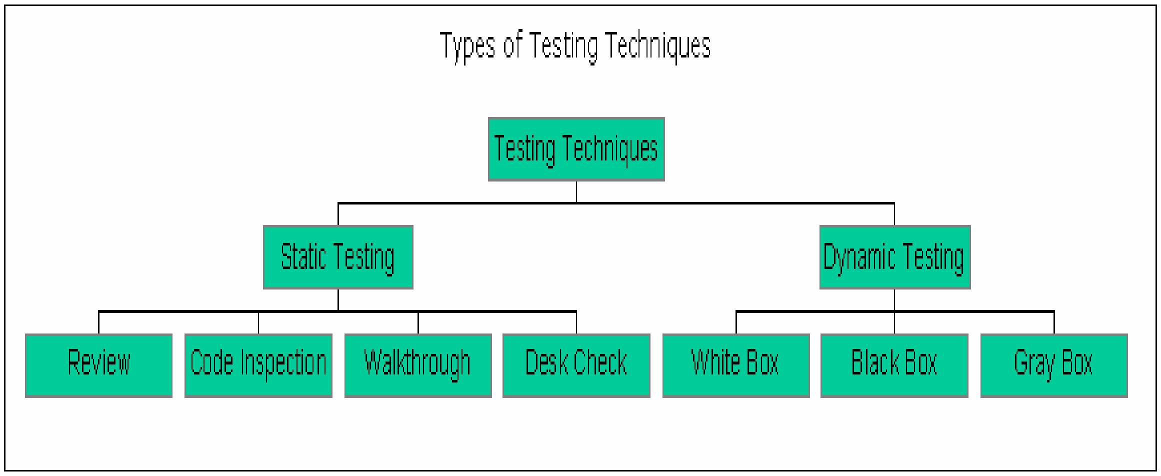
  The FDA is embracing V&V as the primary way of proving your system does what you intended and also meets the needs of the people using it. The resulting V&V effort has become a significant part of the software development effort for a medical device. One of the key pieces to demonstrate that the system is implemented completely is a Requirements Traceability Matrix (RTM), which documents each of the requirements traced to design items, code, unit, integration and system test cases. The RTM is an easy and effective way for documenting your implementation in a manner to which the FDA is familiar - what are the requirements, where are they implemented, and how have you tested them.

TOPIC – 05

Testing Techniques

MANUAL TESTING

01. A Chart of Testing Techniques



02. Static Testing

- ❖ Def :- Testing a software without execution on a computer. Involves just examination/review and evaluation.

- ❖ Static Testing is a process of reviewing the work product and reviewing is done using a checklist.
- ❖ Static Testing helps weed out many errors/bugs at an early stage
- ❖ Static Testing lays strict emphasis on conforming to specifications.
- ❖ Static Testing can discover dead codes, infinite loops, uninitialised and unused variables, standard violations and is effective in finding 30-70% of errors

- ❖ **Static Testing Techniques**
  - ✓ Review
  - ✓ Code Inspection
  - ✓ Walk Through
  - ✓ Desk Checking

03. Dynamic Testing

- ❖ Def :- Dynamic Testing - Testing a software through executing it.

- ❖ **White Box Test Techniques**
  - ✓ Code Coverage
    - Statement Coverage
    - Decision Coverage
    - Condition Coverage
    - Loop Testing
  - ✓ Code complexity
    - Cyclomatic Complexity
  - ✓ Memory Leakage

- **Black Box Test Techniques**
  - ✓ Equivalence Partitioning
  - ✓ Boundary Value Analysis
  - ✓ Error Guessing
  - ✓ Use Case

- **Gray box Testing**

04. Review

- Review is the person dependent activity

- **3 Types of Reviews are :**
  - ✓ Self Review
  - ✓ Peer to Peer Review
  - ✓ Peer Review

- **Code Review Checklist**
  - ✓ Data Reference Errors
  - ✓ Data Declaration Errors
  - ✓ Computation errors
  - ✓ Comparison errors
  - ✓ Control Flow errors
  - ✓ Interface errors
  - ✓ Input/output errors

05. Code Inspection

- Code inspection is a set of procedures and error detection techniques for group code reading.
- Involves reading or visual inspection of a program by a team of people , hence it is a group activity.
- The objective is to find errors but not solutions to the errors
- An inspection team usually consists of:
  - ✓ A moderator
  - ✓ A programmer
  - ✓ The program designer
  - ✓ A test specialist

- **Code Inspection Procedure**
- The moderator distributes the program's listing and design specification to the group well in advance of the inspection session During the inspection
- The programmer narrates the logic of the program, statement by statement
- During the discourse, questions are raised and pursued to determine if errors exist
- The program is analyzed w.r.t a check list of historically common programming errors

- **Code Inspection Helps in**
  - ✓ Detect Defects
  - ✓ Conformance to standards/spec
  - ✓ Requirements Transformation into product

06. Walkthrough

- ❖ Code Walkthrough is a set of procedures and error detection techniques for group reading
- ❖ Like code inspection it is also an group activity
- ❖ In Walkthrough meeting, three to five people are involved. Out of the three, one is **moderator**, the second one is **Secretary** who is responsible for recording all the errors and the third person plays a role of **Test Engineer**

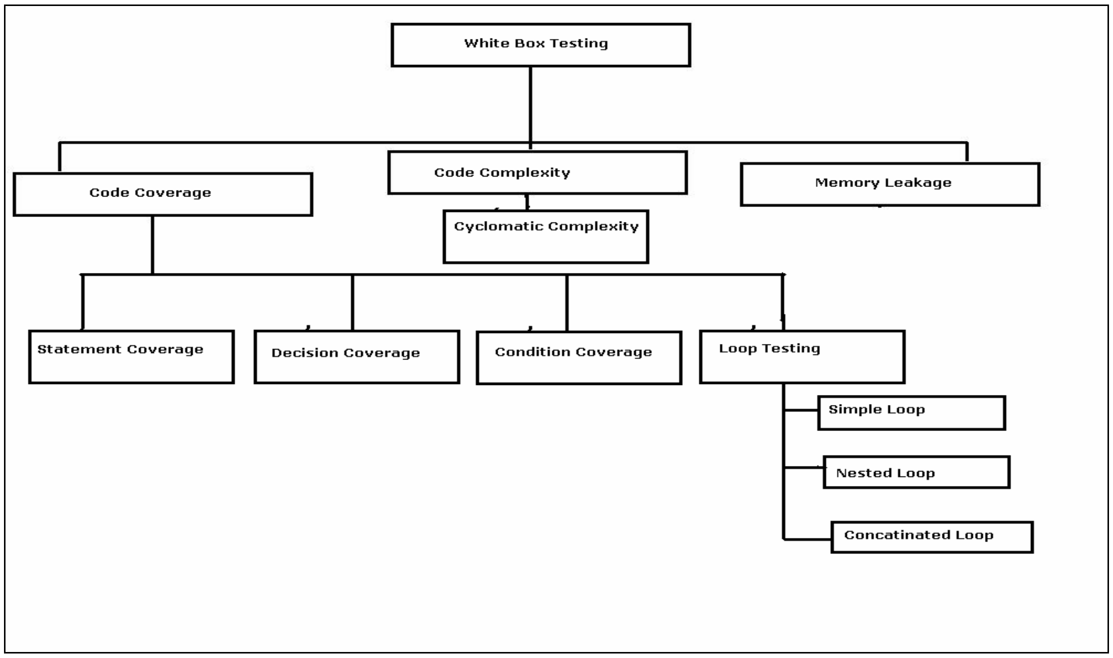
- ❖ **Walkthrough helps in**
  - ✓ Approach to Solution
  - ✓ Find omission of requirements
  - ✓ Style / Concepts Issues
  - ✓ Detect Defects
  - ✓ Educate Team Members

07. Desk Checking

- ❖ Human error detection technique
- ❖ Viewed as a one person inspection or walkthrough
- ❖ A person reads a program and checks it with respect to an error list and/or walks test data through it .

08. White Box Testing

- ❖ White box is logic driven testing and permits Test Engineer to examine the internal structure of the program
- ❖ Examine paths in the implementation
- ❖ Make sure that each statement, decision branch, or path is tested with at least one test case
- ❖ Desirable to use tools to analyze and track Coverage
- ❖ White box testing is also known as **structural**, **glass-box** and **clear-box**.



❖ **White Box Test Techniques**

- ✓ Code Coverage
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Loop Testing
    - Simple Loop
    - Nested Loop
    - Concatenated Loop
- ✓ Code complexity
  - Cyclomatic Complexity
- ✓ Memory Leakage

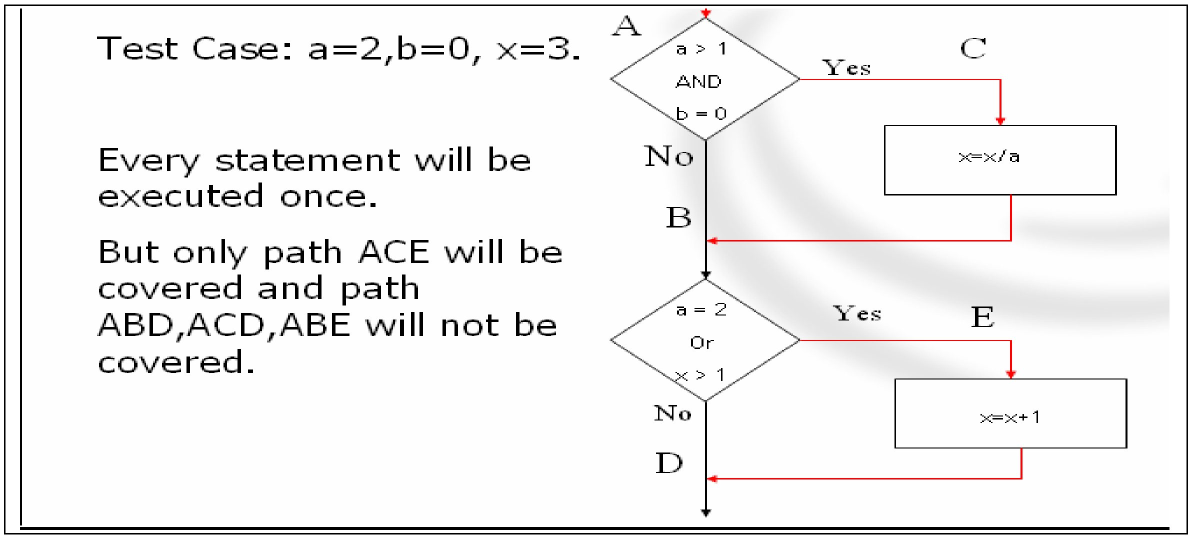
❖ **Code Coverage**

- ✓ Measure the degree to which the test cases exercise or cover the logic (source code) of the program

- ✓ Types
  - Statement Coverage
  - Decision Coverage
  - Conditional Coverage
  - Loop Testing

- ✓ **Statement Coverage**
  - Test cases must be such that all statements in the program is traversed at least once
  - Consider the following snippet of code

```
void procedure(int a, int b, int x)
{    If (a>1) && (b==0)
              { x=x/a;          }
     If (a==2 || x>1)
              {  x=x+1; }
}
```


Test Case: a=2,b=0, x=3.

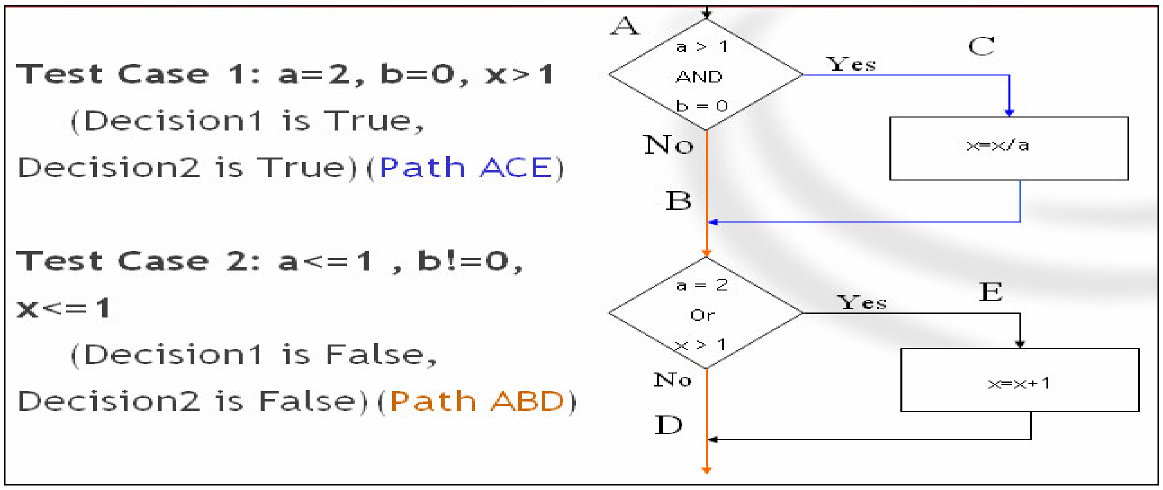
Every statement will be executed once.

But only path ACE will be covered and path ABD,ACD,ABE will not be covered.

- In the above code one test case is only required to execute each of the two if statements at least once:
- Test Case : a=2 , b=0 , x=3
- (Decision1 is True, Decision2 is True)
- However this test case does not help in detecting many of the bugs which may go unnoticed as the false outcomes of the conditions a>1, b=0 & x>1 are not tested.

✓ **Decision Coverage**

- Test cases must be such that each decision has a true and false outcome at least once. If we consider the same example as before, we need at least two test cases to execute the true and false outcome of the decisions at least once.
- **Test Case 1: a=2, b=0, x>1**
  (Decision1 is True, Decision2 is True)(Path ACE)
- **Test Case 2: a<=1, b!=0, x<=1**
  (Decision1 is False, Decision2 is False)(Path ABD)


Test Case 1: a=2, b=0, x>1

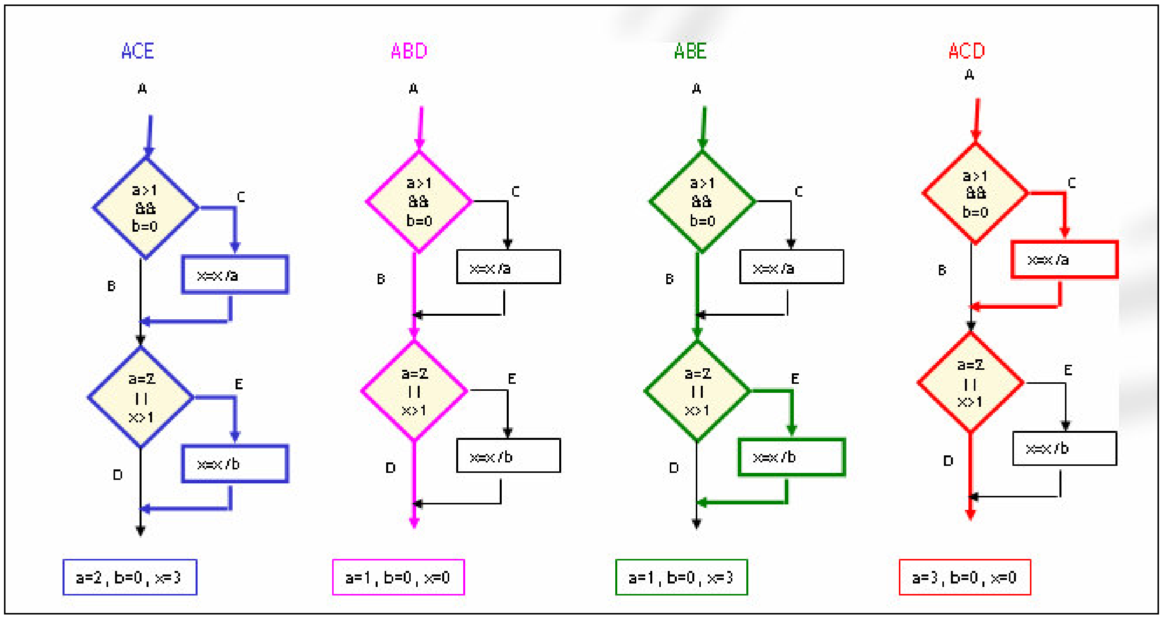
(Decision1 is True, Decision2 is True)(Path ACE)

Test Case 2: a<=1 , b!=0, x<=1

(Decision1 is False, Decision2 is False)(Path ABD)

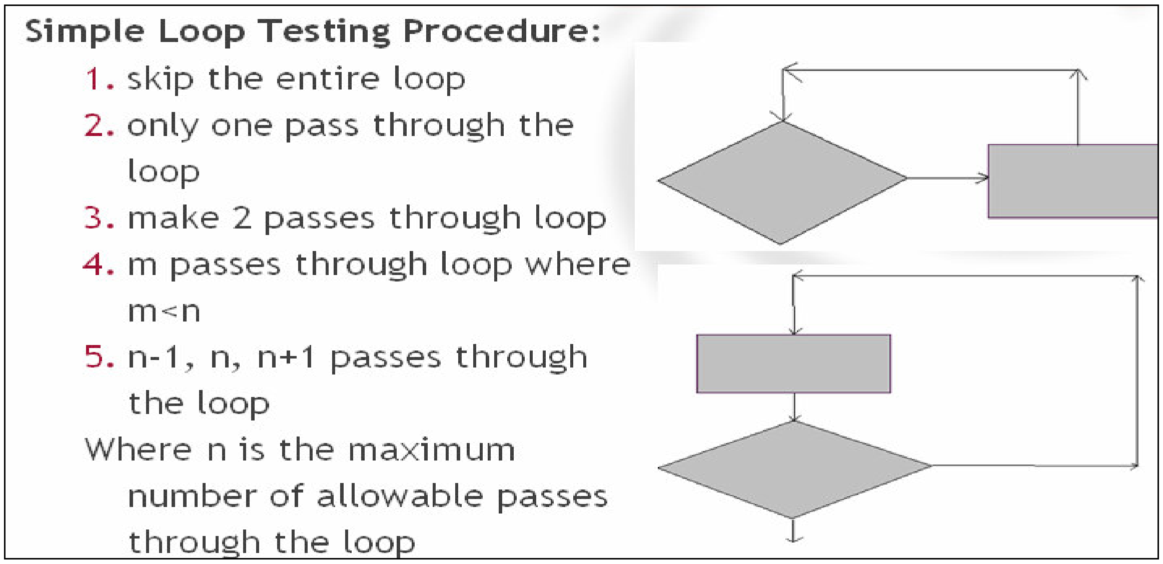
✓ Conditional Coverage

- Test cases are written such that each condition in a decision takes on all possible outcomes at least once.
- **Test Case1 : a=1, b=0, x=3**
- (Condition1 is False,Condition2 is True)(Path ABE)
- **Test Case2: a=1, b=1, x=1**
- (Condition1 is False,Condition2 is False)(Path ABD)



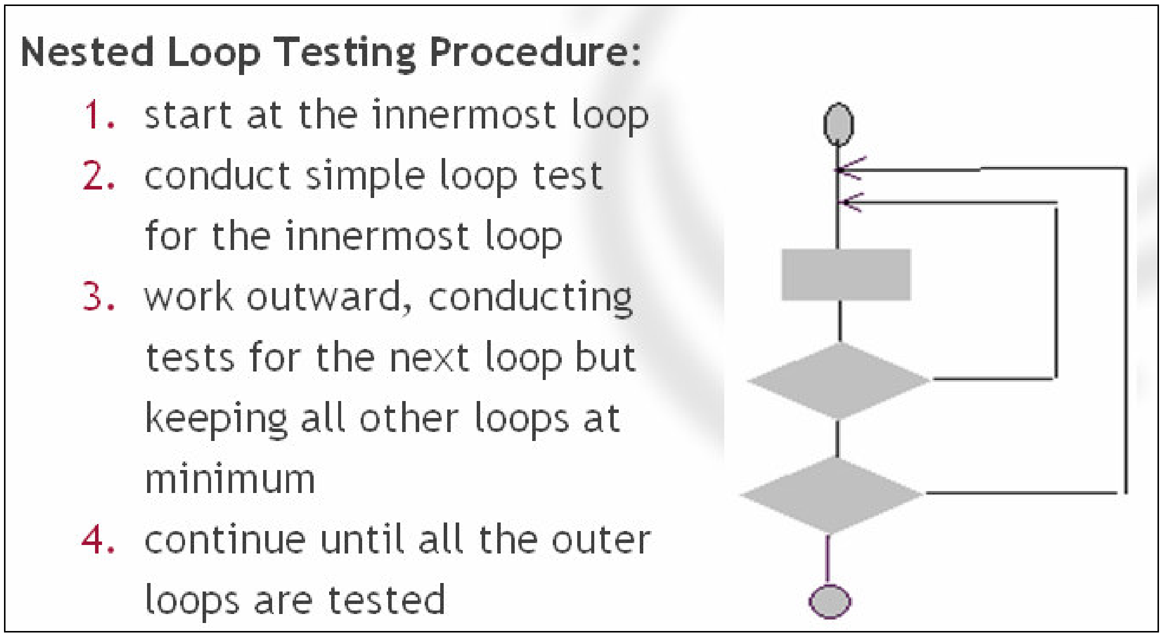
✓ Loop Testing

- Loops testing is a white box testing technique that focuses exclusively on validity of Loop construct
- Types of loops
  - Simple Loop
  - Nested Loop
  - Concatenated Loop


Simple Loop Testing Procedure:

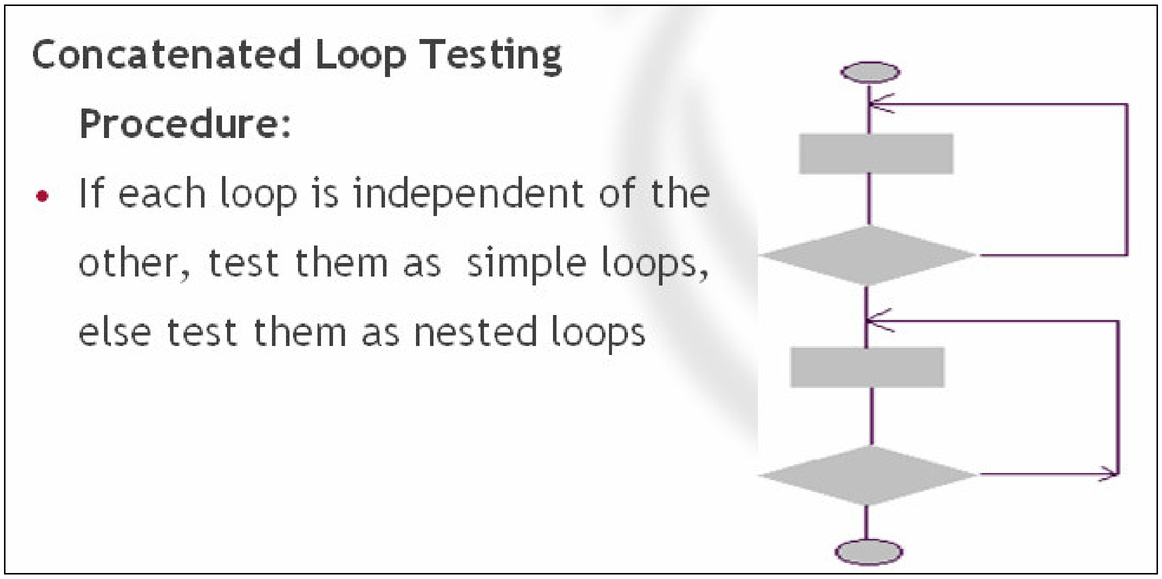
1. skip the entire loop
2. only one pass through the loop
3. make 2 passes through loop
4. m passes through loop where m<n
5. n-1, n, n+1 passes through the loop

Where n is the maximum number of allowable passes through the loop



Nested Loop Testing Procedure:

1. start at the innermost loop
2. conduct simple loop test for the innermost loop
3. work outward, conducting tests for the next loop but keeping all other loops at minimum
4. continue until all the outer loops are tested



Concatenated Loop Testing Procedure:

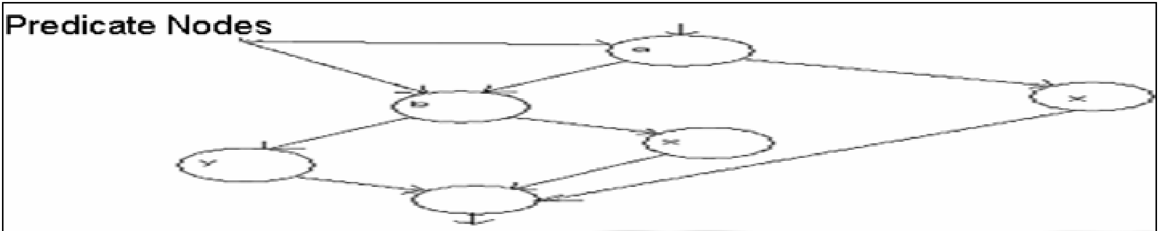
- If each loop is independent of the other, test them as simple loops, else test them as nested loops

❖ **Code complexity**

✓ **Cyclomatic Complexity**

- Cyclomatic Complexity is a software metric that provides a quantitative measure of logical complexity of a program
- When Used in the context of the basis path testing method, value for cyclomatic complexity defines number of independent paths in basis set of a program
- Also provides an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once
- Cyclomatic complexity is often referred to simply as
- program complexity, or as McCabe's complexity

- The cyclomatic complexity of a software module is calculated from a flow graph of the module , when used in context of the basis path testing method
- Cyclomatic Complexity V(G) is calculated one of the three ways:
- **V(G) = E - N + 2** , where E is the number of edges and N = the number of nodes of the graph
- **V(G) = P+1**, where P is the number of predicate nodes
- **V(G) = R** , where number of region in the graph
- E.g.



1. Cyclomatic Complexity, V(G) for a flow Graph G is V(G) = E - N + 2

E = Number of Edges in the graph (7 in the above figure)

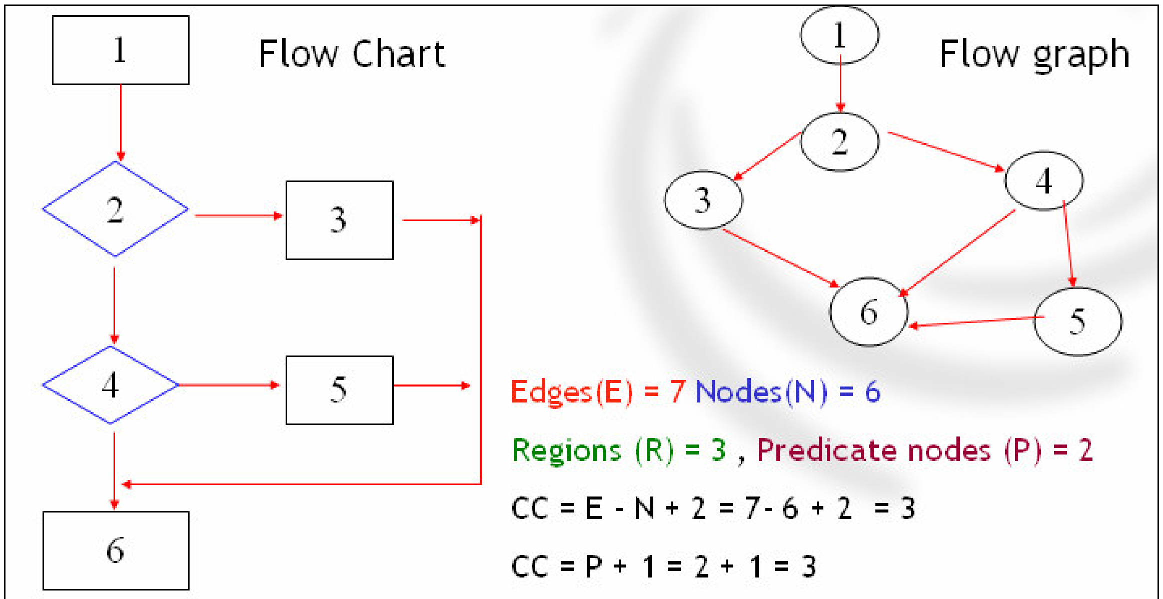
N = number of flow graph Nodes (6)

R = number of Regions (3)

Hence V(G) = 7-6+2 = 3

2. V(G) can also be calculated as V(G) = P+1, where P is the number of predicate nodes. Here V(G) = 2+1 = 3

3. Also V(G) can be calculated as V(G) = R hence V(G) = 3



Edges(E) = 7 Nodes(N) = 6

Regions (R) = 3 , Predicate nodes (P) = 2

CC = E - N + 2 = 7 - 6 + 2 = 3

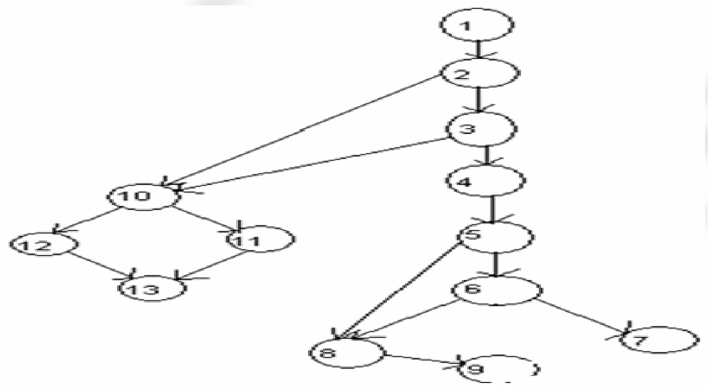
CC = P + 1 = 2 + 1 = 3

CC = R = 3

- **Cyclomatic Complexity – Derive Test Case**

Steps to derive Test Cases from Code

1. Using the Design or code as a foundation, draw a corresponding flow graph.



2. Determine the Cyclomatic Complexity of the resultant flow graph.In the figure drawn in the earlier slide

   V(G) = 15 edges -13 nodes + 2 = 4

3. Determine a basis set of linearly independent paths.In the case of the flow graph drawn paths for e.g. are:

   Path 1:1-2-10-11-13

   Path 2:1-2-10-12-13

   Path 3:1-2-3-10-11-13

The .... At the end of these paths indicate that any path through the remainder of the control structure is acceptable.

4. Prepare test cases that will force the execution on each path in the basis set. This process ensure that all statements have been executed at least once.

Cyclomatic Complexity	Risk Evaluation
1-10	A simple program without much risk
11-20	More Complex, Moderate Risk
21-50	Complex, High Risk Program
Greater than 50	Highly complex,Very high risk program

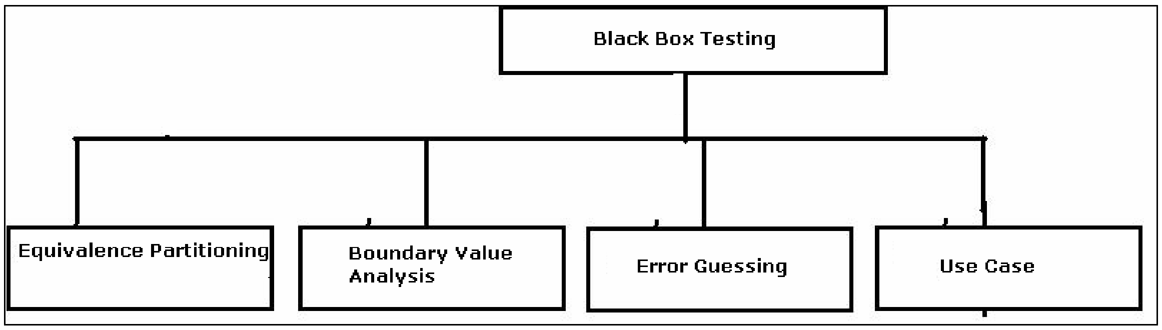
❖ **Memory Leakage**

- ✓ Memory leak is present whenever a program loses track of memory.
- ✓ Memory leaks are most common types of defect and difficult to detect
- ✓ Performance degradation or a deadlock condition occurs
- ✓ Memory leak detection tools help to identify memory allocated but not deallocated
- ✓ Uninitialized memory locations

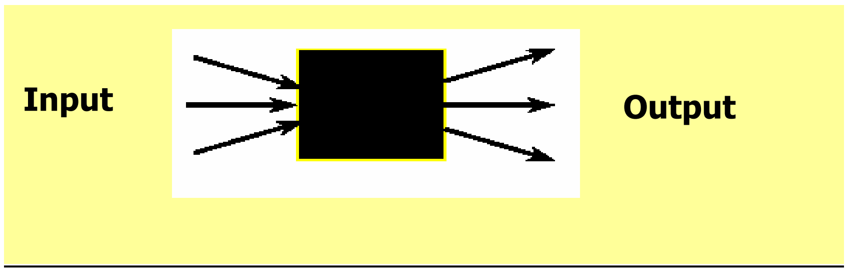
- ✓ **Find the error in the following snippet of code**

```
Void read file (char*);
Void test (bool flag)
{
        char* buf = new char[100];
                if (flag) {
                        read_file(buf);
                        delete [] buf;
                        }
 }
```

09. Black Box Testing



- ❖ Black box is data-driven, or input/output-driven testing
- ❖ The Test Engineer is completely unconcerned about the internal behavior and structure of program
- ❖ Black box testing is also known as **behavioral**, **functional**, **opaque-box** and **closed-box.**



- ❖ Tests are designed to answer the following questions:
- ❖ How is functional validity tested?
- ❖ What classes of input will make good test cases?
- ❖ Is the system particularly sensitive to certain input values?
- ❖ What effect will specific combinations of data have on system operations?

- **Black Box Test Techniques**

  - **Equivalence Partitioning**

    - This method divides the input domain of a program into categories of data for deriving test cases.
    - Identify equivalence classes - the input ranges which are treated the same by the software
    - **Valid classes**: legal input ranges
    - **Invalid classes**: illegal or out of range input values
    - The aim is to group and minimize the number of test cases required to cover these input conditions
    - **Assumption:**
    - If one value in a group works, all will work
    - One from each partition is better than all from one

    - Thus it consists of two steps:
      (01) Identify the Equivalence class
      (02) Write test cases for each class
    - **Example of types of equivalence classes**
      1. If an input condition specifies a continuous range of values, there is one valid class and two invalid classes
      Example: The input variable is a mortgage applicant's income. The valid range is $1000/mo. to $75,000/mo.
      Valid class: {1000 > = income < = 75,000}
      Invalid classes: {income < 1000}, {income > 75,000}

  - **Boundary Value Analysis**

    - "Bugs lurk in corners and congregate at boundaries ….."

    - Boundary Conditions are those situations directly on, above, and beneath the edges of input equivalence classes and output equivalence classes.
    - Boundary value analysis is a test case design technique that complements Equivalence partitioning
    - Test cases at the boundary of each input Includes the values at the boundary , just below the boundary and just above the boundary
    - **Example of Boundary Value Analysis**
      From previous example, we have the valid equivalence class as (1 < *count* < 999).
      Now, according to boundary value analysis,
      we need to write test cases for *count=0, count=1,count=2,count=998,count=999 and count=1000 respectively*
    - If an input condition specifies a range of values A and B, test cases should be designed with values A and B, just above and just below A and B respectively
    - Similarly with a number of values

- **Points to be remembered :**
- We can create 6 test cases using BVA.
- Lower Boundary +1
- Lower Boundary - 1
- Lower Boundary =
- Upper Boundary +1
- Upper Boundary – 1
- Upper Boundary =

✓ **Error Guessing**

- Based on experience and intuition one may add more test cases to those derived by following other methodologies.
- It is an ad hoc approach
- The basis behind this approach is in general people have the knack of "smelling out" errors
- Make a list of possible errors or error-prone situations and then develop test cases based on the list.
- Defects' history are useful. Probability that defects that have been there in the past are the kind that are going to be there in the future.
- Some examples :
- Empty or null lists/strings
- Zero occurrences
- Blanks or null character in strings
- Negative numbers
- ***Example :*** Suppose we have to test the login screen of an application. An experienced test engineer may immediately see if the password typed in the password field can be copied to a text field which may cause a breach in the security of the application.
- Error guessing testing for sorting subroutine situations
  - The input list empty
  - The input list contains only one entry
    - All entries in the list have the same value
    - Already sorted input list

✓ **Use Case**

- Please refer Topic – 6 for Use Cases.

10. Gray Box Testing

- **Definition:** - Gray Box Testing refers to the technique of testing a system with a limited knowledge of the internals of the system. Gray Box testers have access to detailed design documents, with information beyond requirement documents.
- Gray Box tests are generated based on information such as state-based models or architecture diagrams of the target system.

- **Advantages :**

  - ✓ Offers combined benefits – leverages the strengths of both Black Box and White Box testing, whenever possible.
  - ✓ Non-Intrusive – Gray Box does not rely on the access to source code or binaries. Instead, is based on interface definitions, functional specifications and application architecture.
  - ✓ Intelligent Test Authoring – based on the limited information available, a gray box tester can author intelligent test scenarios, especially around data type handling, communication protocols, and exceptional handling.
  - ✓ Unbiased Testing – the demarcation between testers and developers is still maintained. The handoff is only around interface definitions and documentation, without access to source code or binaries.

- **Disadvantages :**

  - ✓ Partial code Coverage – since the source code or binaries are not available, the ability to traverse code paths is still limited by the tests deduced through available information. The coverage depends on the tester authoring skills.

====**Topic - 5**=== XXX === **Testing Techniques – Static & Dynamic** ===== XXX ==

TOPIC – 06

USE CASES

MANUAL TESTING

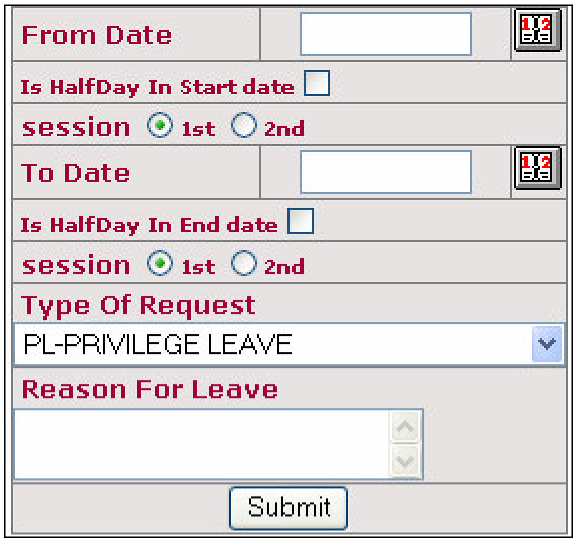
01. Online Leave Application

❖ **Requirements :**

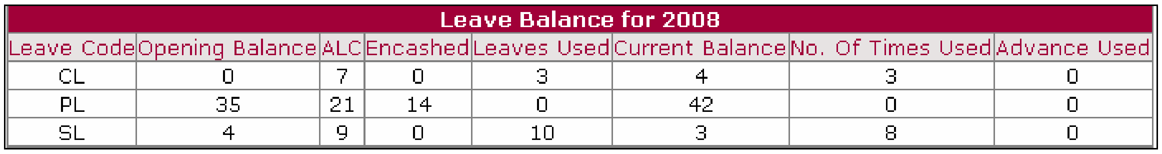
Pre Condition: User is employee of company and having access to Intranet
Description: User wants to register for Leave (1/2 day or full day)

1. User will access Intranet
2. Click on Attendance System
3. Select Leave Application – Leave application form will be displayed (Diagram 1)
4. Select From Date (Mandatory)
5. Check Half Day on start date check box (Optional for full day leave)
6. Select session (1st session or 2nd session) radio button (Default is session 1)
7. Select To Date (Mandatory)
8. Check Half Day on end date check box (Optional for full day leave)
9. Select session (1st session or 2nd session) radio button (Default is session 1)
10. Select Type of Leave from the list provided (Mandatory) (Types of leave - CL, PL, SL)
11. Enter reason for leave (Mandatory)
12. Click on Submit to apply for the leave.

❖ **Screen shots :**



From Date	
Is HalfDay In Start date ☐	
session [✓ 1st] 2nd	
To Date	
Is HalfDay In End date ☐	
session [✓ 1st] 2nd	
Type Of Request	PL-PRIVILEGE LEAVE
Reason For Leave	



Leave Balance for 2008

Leave Code	Opening Balance	ALC	Encashed	Leaves Used	Current Balance	No. Of Times Used	Advance Used
CL	0	7	0	3	4	3	0
PL	35	21	14	0	42	0	0
SL	4	9	0	10	3	8	0

- **Use Case :**

Use Case ID:	UC_01		
Use Case Name:	Online Leave Registration		
Created By:	Prajesh J. Kansara	**Reviewed By**	
Date Created:	25/07/2008	**Date Reviewed**	

Scenario / Goal	Online registration of the Leave of the employee.
Assumptions:	1. The user is the employee of company. 2. The user has internet access. 3. The user has sufficient number of leave to apply.
Reference:	Leave application_document.doc Leave application.doc
Includes:	-- NA --
Description:	User wants to register his leave in the Leave application System.
Primary Actor	The employee of company.
Precondition:	User is employee of company and having access to internet.
Basic Flow	**Register Leave Application for one full day.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify the Full day. 4. Specify the valid To Date. 5. Specify the valid Type of Leave. 6. Specify Reason for Leave. 7. Apply for Leave.
Post conditions:	System Generated Message **Msg001**
Precondition:	For half day leave From Date and To Date must be same.
Alternate Flow-1	**Register Leave Application for Half day with 1st session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 1st Session for From Date. 5. Specify the To Date same as From Date. 6. Specify the valid Type of Leave. 7. Specify Reason for Leave. 8. Apply for Leave.
Post conditions:	System Generated Message **Msg002**
Precondition:	For half day leave From Date and To Date must be same.
Alternate Flow-2	**Register Leave Application for Half day with 2nd session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 2nd Session for From Date. 5. Specify the To Date same as From Date. 6. Specify the valid Type of Leave.

	7. Specify Reason for Leave. 8. Apply for Leave.
Post conditions:	System Generated Message **Msg002**
Alternate Flow-3	**Register Leave Application for One day with From Date 1st Session and To Date 1st Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 1st Session for From Date. 5. Specify the valid To Date. 6. Specify Half day. 7. Specify 1st Session for To Date. 8. Specify the valid Type of Leave. 9. Specify Reason for Leave. 10. Apply for leave.
Post conditions	System Generated Message **Msg003**
Alternate Flow-4	**Register Leave Application for One day with From Date 1st Session and To Date 2nd Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 1st Session for From Date. 5. Specify the valid To Date. 6. Specify Half day. 7. Specify 2nd Session for To Date. 8. Specify the valid Type of Leave. 9. Specify Reason for Leave. 10. Apply for leave.
Post conditions:	System Generated Message **Msg003**
Alternate Flow-5	**Register Leave Application for One day with From Date 2nd Session and To Date 1st Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 2nd Session for From Date. 5. Specify the valid To Date. 6. Specify Half day. 7. Specify 1st Session for To Date. 8. Specify the valid Type of Leave. 9. Specify Reason for Leave. 10. Apply for Leave.
Post conditions:	System Generated Message **Msg003**
Alternate Flow-6	**Register Leave Application for One day with From Date 2nd Session and To Date 2nd Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 2nd Session for From Date. 5. Specify the valid To Date.

	6. Specify Half day. 7. Specify 2nd Session for To Date. 8. Specify the valid Type of Leave. 9. Specify Reason for Leave. 10. Apply for Leave.
Post conditions:	System Generated Message **Msg003**
Alternate Flow-7	**Register Leave Application for One and half day with full From Date and To Date 1st Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify the valid To Date. 4. Specify Half day. 5. Specify 1st Session for To Date. 6. Specify the valid Type of Leave. 7. Specify Reason for Leave. 8. Apply for Leave.
Post conditions:	System Generated Message **Msg004**
Alternate Flow-8	**Register Leave Application for One and half day with full From Date and To Date 2nd Session.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify the valid To Date. 4. Specify Half day. 5. Specify 2nd Session for To Date. 6. Specify the valid Type of Leave. 7. Specify Reason for Leave. 8. Apply for Leave.
Post conditions:	System Generated Message **Msg004**
Alternate Flow-9	**Register Leave Application for One and half day with From Date 1st Session and full To Date.** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 1st Session for From Date. 5. Specify the valid To Date. 6. Specify the valid Type of Leave. 7. Specify Reason for Leave. 8. Apply for Leave.
Post conditions:	System Generated Message **Msg004**
Alternate Flow-10	**Register Leave Application for One and half day with From Date 2nd Session and full To Date** 1. User invokes the application. 2. Specify the valid From Date. 3. Specify Half day. 4. Specify 2nd Session for From Date. 5. Specify the valid To Date. 6. Specify the valid Type of Leave. 7. Specify Reason for Leave. 8. Apply for Leave.

Post conditions:	System Generated Message **Msg004**
Exceptional Flow-1	User is not an employee.
Post conditions:	System Generate Error Message **Err001**
Exceptional Flow-2	User is not able to access the internet.
Post conditions:	System Generate Error Message **Err001**
Exceptional Flow-3	1. User invokes the application. 2. Specify the invalid From Date. 3. Specify the valid To Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave.
Post conditions:	System Generate Error Message **Err002**
Exceptional Flow-4	1. User invokes the application. 2. Specify the valid From Date. 3. Specify the invalid To Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave.
Post conditions:	System Generate Error Message **Err003**
Exceptional Flow-5	1. User invokes the application. 2. Specify the valid From Date. 3. Specify the valid To Date. 4. Specify the invalid Type of Leave. 5. Specify Reason for Leave. 6. Apply for leave.
Post conditions:	System Generate Error Message **Err004**
Exceptional Flow-6	1. User invokes the application. 2. Specify the valid From Date. 3. Specify the valid To Date. 4. Specify the valid Type of Leave. 5. Not specify Reason for Leave. 6. Apply for Leave.
Post conditions:	System Generate Error Message **Err005**
Exceptional Flow-7	1. User invokes application. 2. Specify valid From Date. 3. Specify To Date as less than From Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave
Post conditions:	System Generate Error Message **Err006**
Precondition:	Already register for the leave application by the user.
Exceptional Flow-8	1. User invokes the application. 2. Specify valid From Date. 3. Specify To Date as less than From Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave.
Post condition:	System Generate Error Message **Err007**
Precondition:	Leave application for holiday.

Exceptional Flow-9	1. User invokes the application. 2. Specify valid From Date. 3. Specify To Date as less than From Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave.
Post Condition:	System Generate Error Message **Err008**
Precondition:	No leave remain in account.
Exceptional Flow-10	1. User invokes the application. 2. Specify valid From Date. 3. Specify To Date as less than From Date. 4. Specify the valid Type of Leave. 5. Specify Reason for Leave. 6. Apply for Leave.
Post Condition:	System Generate Error Message **Err009**

Err001	**User is not able to apply for leave.**
Err002	**Invalid From Date.**
Err003	**Invalid To Date.**
Err004	**Invalid Type of leave.**
Err005	**Reason must be specified.**
Err006	**From Date should not greater than To Date.**
Err007	**This application already registers by you.**
Err008	**You are applying for leave on holiday.**
Err009	**In your account no leave remain.**

Msg001	**The employee is registered for the full day leave.**
Msg002	**The employee is registered for the half day leave.**
Msg003	**The employee is registered for the one day leave.**
Msg004	**The employee is registered for the one and half day leave.**

02. Online Course Registration

Goal:	Online registration for a course
Description:	Registers a student for a course.
Primary Actor(s):	A student of the Institute.
Pre-Conditions:	The user is qualified as per the prerequisites for the course. (eg. To register for course ABC, the minimum educational qualification requirement is that the student should be a BE.)
Basic Flow:	1. User enters the valid Student Id and valid password. 2. User specifies valid course for registration. 3. User enters his/her educational qualification. 4. User chooses the option to Register.
Post-Conditions:	The student is registered for the course and the Registration Number is generated.
Alternate Flow(s):	1. User enters the valid Student Id and Password. User specifies the course for which to register. User enters his/her educational qualification. User saves the data for later registration and quits. Post-condition: The student details are saved for later use. 2. User specifies invalid Student Id and valid Password. User specifies valid course and qualification. User chooses the option to Register. Post-condition: Error 1 3. User enters valid Student Id and invalid Password. User specifies valid course and qualification. User chooses the option to Register. Post-condition: Error 1 4. User enters the valid Student Id and valid Password. User specifies valid course for registration. User enters his/her educational qualification. (User does not meet the qualifying criteria for the selected course.) User chooses the option to Register. Post-condition: Error 2 5. User enters the valid Student Id and valid password. User specifies valid course for which the last date of registration occurs in the past. User enters his/her educational qualification. User chooses the option to Register. Post-condition: Error 3 6. User enters the valid Student Id and valid password. User specifies valid course for which capacity is already full. User enters his/her educational qualification. User chooses the option to Register. Post-condition: Error 4
Exceptions / Error Handling / Warnings:	1. Unidentified Student. 2. Cannot enroll, Unfulfilled Pre-requisites. 3. Cannot enroll, Course Registration closed. 4. Cannot enroll, Course Full.

03. Use Case Checklist

1.	Is each concrete use case involved with at least one actor? If not, something is wrong; a use case that does not interact with an actor is superfluous, and you should remove it
2.	Is each use case independent of the others? If two use cases are always activated in the same sequence, you should probably merge them into one use case
3.	For an included use case: does it make assumptions about the use cases that include it? Such assumptions should be avoided, so that the included use case is not affected by changes to the including use cases
4.	Do any use cases have very similar behaviors or flows of events? If so - and if you wish their behavior to be similar in the future - you should merge them into a single use case. This makes it easier to introduce future changes. Note: you must involve the users if you decide to merge use cases, because the users, who interact with the new, merged use case will probably be affected
5.	Has part of the flow of events already been modeled as another use case? If so, you can have the new use case use the old one.
6.	Is some part of the flow of events already part of another use case? If so, you should extract this sub flow and have it be used by the use cases in question. Note: you must involve the users if you decide to "reuse" the sub flow, because the users of the existing use case will probably be affected
7.	Should the flow of events of one use case be inserted into the flow of events of another? If so, you model this with an extend-relationship to the other use case
8.	Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage? If not, you change their names
9.	Do customers and users alike understand the names and descriptions of the use cases? Each use-case name must describe the behavior the use case supports.
10.	Does the use case meet all the requirements that obviously govern its performance? You must include any (nonfunctional) requirements to be handled in the object models in the use-case *Special Requirements*
11.	Does the communication sequence between actor and use case conform to the user's expectations?
12.	Is it clear how and when the use case's flow of events starts and ends?
13.	Behavior might exist that is activated only when a certain condition is not met. Is there a description of what will happen if a given condition is not met?
14.	Are any use cases overly complex? If you want your use-case model to be easy to understand, you might have to split up complex use cases.
15.	Does a use case contain disparate flows of events? If so, it is best to divide it into two or more separate use cases. A use case that contains disparate flows of events will be very difficult to understand and to maintain
16.	Is the subflow in a use case modeled accurately?
17.	Is it clear who wishes to perform a use case? Is the purpose of the use case also clear?
18.	Are the actor interactions and exchanged information clear?
19.	Does the brief description give a true picture of the use case?

04. Use Case to Test Case

Use Cases to Test Cases – Online registration for a course

Test Case ID	Scenario	Actions	Values	Error Messages / Warnings / Information Messages	Expected Results
Tc01	Scenario 1– Basic flow (Successful Registration)	Enter valid Student Id.	Xyz12		The student is registered for the course and the Registration No. is displayed.
		Enter valid Password	Abc123		
		Select valid Course.	M101		
		Enter Educational qualification.	B.E.		
		Register for the course.			
Tc02	Scenario 2 – Alternate flow 1 (Saved for later registration)	Enter valid Student Id.	Xyz12		The student details are saved for later use.
		Enter valid Password	Abc123		
		Select valid Course.	M102		
		Enter valid Educational qualification.	B.E.		
		Save the data for later registration.			
Tc03	Scenario 3 – Alternate flow 2 (Unidentified Student)	Enter Invalid Student Id.	Xyz	Error – Unidentified student.	
		Register for the course.			
Tc04	Scenario 4 – Alternate flow 3 (Unidentified Student)	Enter valid Student Id.	Xyz12	Error – Unidentified student	
		Enter invalid password.	Abc		
		Register for the course.			
Tc05	Scenario 5 – Alternate flow 4 (Pre-requisite not fulfilled)	Enter valid Student Id.	Xyz12	Error – Cannot enroll, Unfulfilled pre-requisite.	
		Enter valid Password	Abc123		
		Select valid Course.	M101		
		Enter invalid Educational qualification.	H.S.C.		
		Register for the course.			
Tc06	Scenario 6 – Alternate flow 5 (Course Registration closed)	Registration date (Date of test execution) greater than last date of Registration for course E203.		Error – Cannot enroll, Course Registration closed.	
		Enter valid Student Id.	Xyz12		
		Enter valid Password	Abc123		
		Select valid Course.	E203		
		Enter valid Educational qualification.	B.E.		
		Register for the course.			
Tc07	Scenario 7 – Alternate flow 6 (Course Full)	Course K502 is already full.		Error – Cannot enroll, Course full.	
		Enter valid Student Id.	Xyz12		
		Enter valid Password	Abc123		
		Select valid Course.	K502		
		Enter valid Educational qualification.	B.E.		
		Register for the course.			

====**Topic - 6**=== XXX ===**USE CASES**===== XXX =====================

TOPIC – 07

TEST CASES

01. Format of test cases

Test Case ID	Pre-Condition	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result	Actual Result	Status (Pass/ Fail)

02. Create test cases to validate the Date Filed

- Validate Date Filed using Black Box Testing Techniques. Format of Date field is dd/mm/yyyy

TEST CASES

Module Name: Validate Date Field using Black Box Testing Techniques. Format of Date field is dd/mm/yyyy
Producer: Prajesh Kansara
Produced Date :25/07/2008

Assumptions : Validation for Year is from 1900 To 2050

Test Case ID	Pre-Cond	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result
TC_1		To make the date field blank	Keep the date field blank		An error message should be displayed (Error ID - 1)
TC_2		To verify the date when no separators are used	Entering date without any separators	26122006	An error message should be displayed (Error ID - 6)
TC_3		To verify the date when different separators are used	Entering date with separators other then (/)	26#12#2006	An error message should be displayed (Error ID - 6)
TC_4		To verify the date when different formats are used	Entering date with other format like mm/dd/yyyy or yyyy/mm/dd	12/26/2006 2006/12/26	An error message should be displayed (Error ID - 6)
TC_5		To verify the date when day part of date is entered other than numeric value	Entering day other than numeric value with valid month and year in date	five/12/2006 c$/12/2006	An error message should be displayed (Error ID - 2)
TC_6		To verify the date when day part of date is blank	Entering day as blank with valid month and year in date	/12/2006	An error message should be displayed (Error ID - 3)
TC_7		To verify the date when day part of date is negative	Entering day as negative number with valid month and year in date	-26/12/2006	An error message should be displayed (Error ID - 3)
TC_8		To verify the date when day part of date is not in two digit	Entering day less than or more than 2 digits with valid month and year in date	1/12/2006 111/12/2006	An error message should be displayed (Error ID - 3)

TC_9		To verify the date with lower boundary for day part of date	Entering day < 1 with valid month and year in date	00/12/2006	An error message should be displayed (Error ID - 3)
			Entering day = 1 with valid month and year in date	01/12/2006	Date is accepted
			Entering day > 1 with valid month and year in date	02/12/2006	Date is accepted
TC_10		To verify the date with upper boundary for day part of date (For Jan, March, May, July, Augest, October, December)	Entering day > 31 with valid month and year in date	32/12/2006	An error message should be displayed (Error ID - 3)
			Entering day = 31 with valid month and year in date	31/12/2006	Date is accepted
			Entering day < 31 with valid month and year in date	30/12/2006	Date is accepted
TC_11		To verify the date with upper boundary for day part of date (For Feb, April, June, Sep, Nov)	Entering day > 30 with valid month and year in date	31/04/2006	An error message should be displayed (Error ID - 3)
			Entering day = 30 with valid month and year in date	30/04/2006	Date is accepted
			Entering day < 30 with valid month and year in date	29/04/2006	Date is accepted
TC_12		To verify the date when invalid day part for the February	Entering day more than 28 for February month for none leap year	29/02/2006	An error message should be displayed (Error ID - 6)
			Entering day 29 for February month for leap year	29/02/1900	Date is accepted
TC_13		To verify the date when month part of date is other than numeric value	Entering month other than numeric value with valid day and year in date	26/five/2006 26/c^/2006	An error message should be displayed (Error ID - 2)
TC_14		To verify the date when month part of date is blank	Entering month as blank with valid day and year in date	26/ /2006	An error message should be displayed (Error ID - 4)

TC_15		To verify the date when month part of date is negative	Entering month as negative number with valid day and year in date	26/-12/2006	An error message should be displayed (Error ID - 4)
TC_16		To verify the date when month part of date is not in two digit	Entering month less than or more than 2 digits with valid month and year in date	26/2/2006 26/222/2006	An error message should be displayed (Error ID - 4)
TC_17		To verify the date for lower boundary for month part	Entering month < 1 with valid month and year in date	26/00/2006	An error message should be displayed (Error ID - 4)
			Entering month = 1 with valid month and year in date	26/01/2006	Date is accepted
			Entering month > 1 with valid month and year in date	26/02/2006	Date is accepted
TC_18		To verify the date for upper boundary for month parthen month part	Entering month > 12 with valid month and year in date	26/13/2006	An error message should be displayed (Error ID - 4)
			Entering month = 12 with valid month and year in date	26/12/2006	Date is accepted
			Entering month < 12 with valid month and year in date	26/11/2006	Date is accepted
TC_19		To verify the date when year part of date is other than numeric value	Entering year other than numeric value with valid day and month in date	26/12/six 26/12/%*%*	An error message should be displayed (Error ID - 2)
TC_20		To verify the date when year part of date is blank	Entering year as blank with valid day and month in date	26/12/	An error message should be displayed (Error ID - 5)
TC_21		To verify the date when year part of date is negative	Entering year as negative number with valid day and year in date	26/12/-2006	An error message should be displayed (Error ID - 5)
TC_22		To verify the date when year part of date is not in four digit	Entering year less than or more than 4 digits with valid month and year in date	26/12/06 26/12/20006	An error message should be displayed (Error ID - 5)

TC_23		To verify the date for lower boundary of year part	Entering year < 1900 with valid day and month	26/12/1899	An error message should be displayed (Error ID - 5)
			Entering year = 1900 with valid day and month	26/12/1900	Date is accepted
			Entering year > 1900 with valid day and month	26/12/1901	Date is accepted
TC_24		To verify the date for upper boundary of year part	Entering year > 2050 with valid day and month	26/12/2051	An error message should be displayed (Error ID - 5)
			Entering year = 2050 with valid day and month	26/12/2050	Date is accepted
			Entering year > 2050 with valid day and month	26/12/2051	Date is accepted
TC_25		To verify the valid date	Entering valid date	26/12/2006 31/10/2006 31/03/2006 31/12/2006 30/01/2006 30/03/2006 30/11/2006 30/12/2006 29/02/1900 29/02/1904	Date is accepted

Error Message Table

Error ID	Error Description	Error ID	Error Description
1	Please enter date	5	Invalid year in date
2	Enter only digits for date	6	Invalid date
3	Invalid day in date		
4	Invalid month in date		

03. Create test cases to validate the Phone No. Field

- Validate Phone Number field.
  Format of the number is 999-99999-99999999
  - Country Code (10 to 999)
  - City Code (10 to 99999)
  - Phone Number (1000000 to 99999999)

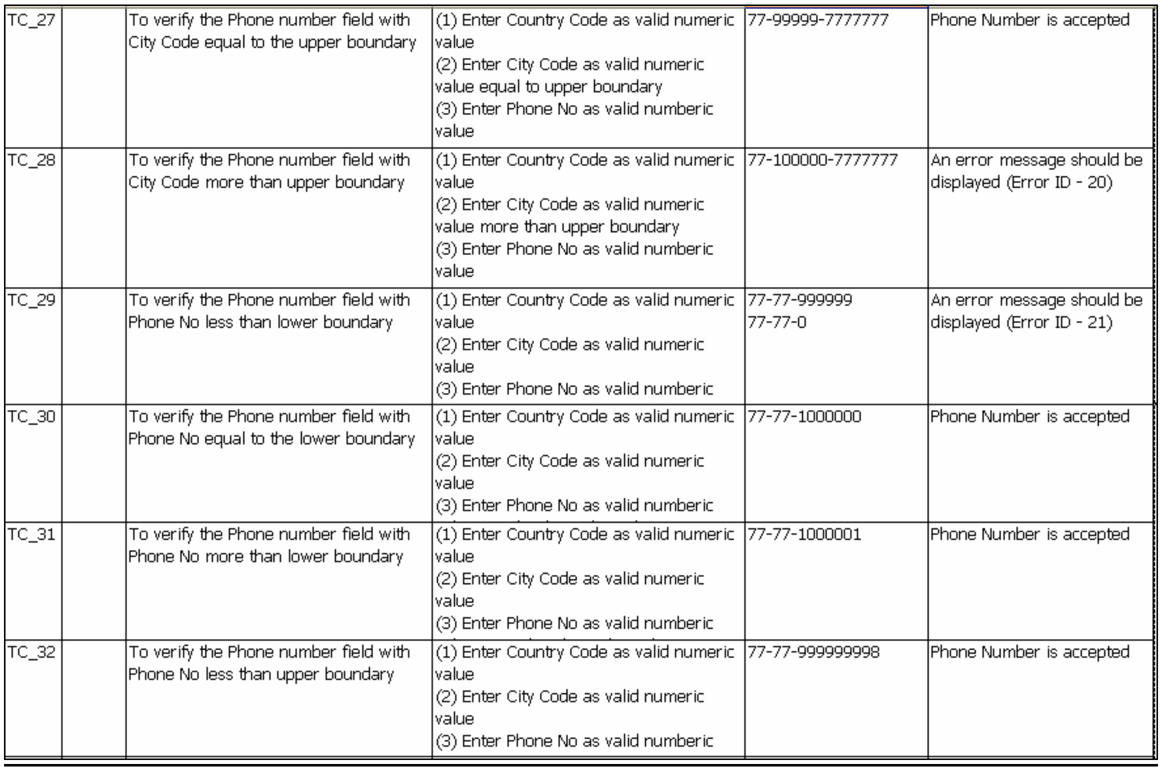
TEST CASES

Module Name: Validate Phone Number Field. Format of the numbers Country Code (10-999), City Code (10-99999) and Phone No (1000000-999999999)
Producer: Prajesh Kansara
Produced Date :25/07/2008

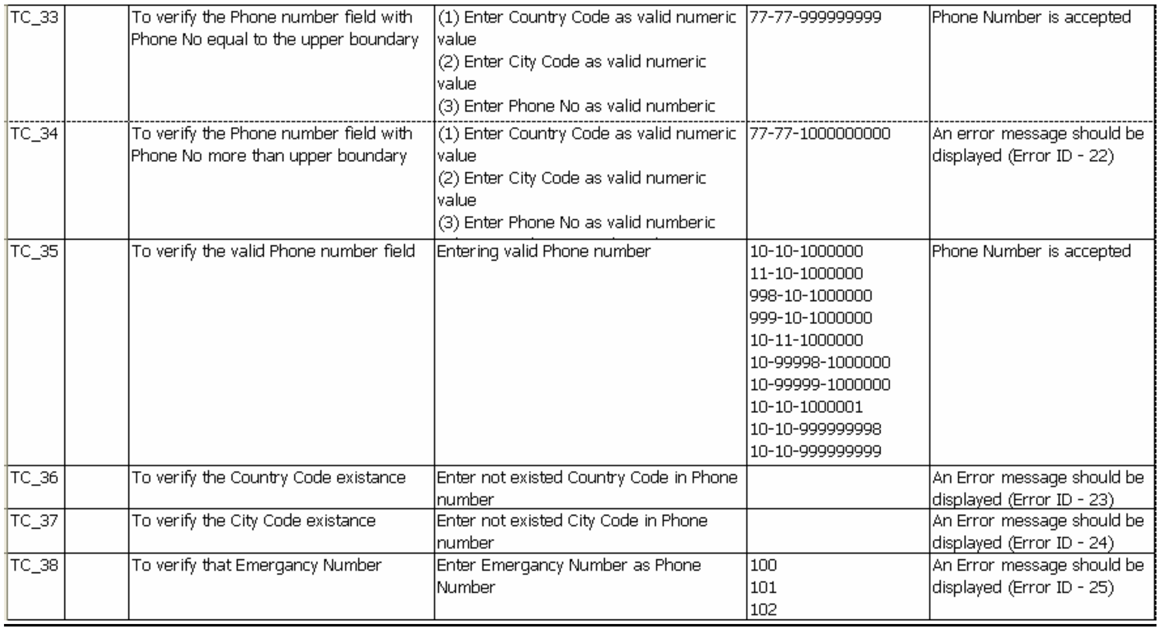
Assumptions : In the Phone Number field Country Code, City Code and Phone No separated with (-) i.e. 77-77-777777777

Test Case ID	PreCond	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result
TC_1		To make the Phone number is blank or not	Keep the Phone No field		An error message should be displayed (Error ID - 1)
TC_2		To verify the Phone number field without entering Country Code	Enter Phone Number without Country Code	-77-7777777	An error message should be displayed (Error ID - 2)
TC_3		To verify the Phone number field without entering City Code	Enter Phone Number without City Code	77- -7777777	An error message should be displayed (Error ID - 3)
TC_4		To verify the Phone number field without entering Phone No	Enter Phone Number without Phone No	77-77-	An error message should be displayed (Error ID - 4)
TC_5		To verify the Phone number field without entering City Code and Phone	Enter Phone Number without City Code and Phone No	77- -	An error message should be displayed (Error ID - 5)
TC_6		To verify the Phone number field without entering Country Code and	Enter Phone Number without Country Code and Phone No	-77-	An error message should be displayed (Error ID - 6)
TC_7		To verify the Phone number field without entering Country Code and City	Enter Phone Number without Country Code and City Code	- -7777777	An error message should be displayed (Error ID - 7)
TC_8		To verify the Phone number field without entering Country Code, City Code and Phone No	Enter Phone Number without Country Code, City Code and Phone No	- -	An error message should be displayed (Error ID - 8)
TC_9		To verify the Phone number field with other format	Enter Phone Number with different separator (i.e. #,@,%,*,=,:)	77#77#7777777	An error message should be displayed (Error ID - 9)
TC_10		To verify the Phone number field with Country Code other than number	(1) Enter Country Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z], space) (2) Enter valid City Code (3) Enter valide Phone No	ch-77-7777777 ##-77-7777777 c#-77-7777777	An error message should be displayed (Error ID - 10)
TC_11		To verify the Phone number field with City Code other than number	(1) Enter valid Country Code (2) Enter City Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z], space)	77-ch--7777777 77-##-7777777 77-c#-7777777	An error message should be displayed (Error ID - 11)
TC_12		To verify the Phone number field with Phone No other than number	(1) Enter valid Country Code (2) Enter valid City Code (3) Enter Phone No other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-	77-77-chchchchch 77-77-######### 77-77-c#c#c#h#h#	An error message should be displayed (Error ID - 12)
TC_13		To verify the Phone number field with Country Code and City Code other than number	(1) Enter Country Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z],space) (2) Enter City Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-	ch-ch-7777777 ##-@@-7777777 c#-h@-7777777	An error message should be displayed (Error ID - 13)
TC_14		To verify the Phone number field with Country Code and Phone No other than number	(1) Enter Country Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z],space) (2) Enter valie City code (3) Enter Phone No other then numeric	ch-77-chchchchc ##-77-@@@@@@@ c#-77-c#c#h@h@h	An error message should be displayed (Error ID - 14)

TC_15		To verify the Phone number field with City Code and Phone No other than number	(1) Enter valid Country code (2) Enter City Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z],space) (3) Enter Phone No other then numeric	77-ch--chchchchc 77-##-@@@@@@@ 77-c#-c#c#h@h@h	An error message should be displayed (Error ID - 15)
TC_16		To verify the Phone number field with Country Code, City Code and Phone No other than number	(1) Enter Country Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z],space) (2) Enter City Code other then numeric value (i.e. #,@,%,*,=,:,[A-Z][a-z],space) (3) Enter Phone No other then numeric	c#-h#-c#h#c#h#	An error message should be displayed (Error ID - 16)
TC_17		To verify the Phone number field with Country Code less than lower boundary	(1) Enter Country Code as valid numeric value less than lower boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	9-77-7777777 0-77-7777777	An error message should be displayed (Error ID - 17)
TC_18		To verify the Phone number field with Country Code equal to the lower boundary	(1) Enter Country Code as valid numeric value equal to lower boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	10-77-7777777	Phone Number is accepted
TC_19		To verify the Phone number field with Country Code more than lower boundary	(1) Enter Country Code as valid numeric value more than lower boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	11-77-7777777	Phone Number is accepted
TC_20		To verify the Phone number field with Country Code less than upper boundary	(1) Enter Country Code as valid numeric value less than upper boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	998-77-7777777	Phone Number is accepted
TC_21		To verify the Phone number field with Country Code equal to the upper boundary	(1) Enter Country Code as valid numeric value equal to upper boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	999-77-7777777	Phone Number is accepted
TC_22		To verify the Phone number field with Country Code more than upper boundary	(1) Enter Country Code as valid numeric value more than upper boundary (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic value	1000-77-7777777	An error message should be displayed (Error ID - 18)
TC_23		To verify the Phone number field with City Code less than lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value less than lower boundary (3) Enter Phone No as valid numberic value	77-9-7777777 77-0-7777777	An error message should be displayed (Error ID - 19)
TC_24		To verify the Phone number field with City Code equal to the lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value equal to lower boundary (3) Enter Phone No as valid numberic value	77-10-7777777	Phone Number is accepted
TC_25		To verify the Phone number field with City Code more than lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value more than lower boundary (3) Enter Phone No as valid numberic value	77-11-7777777	Phone Number is accepted
TC_26		To verify the Phone number field with City Code less than upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value less than upper boundary (3) Enter Phone No as valid numberic	77-99998-7777777	Phone Number is accepted



TC_27		To verify the Phone number field with City Code equal to the upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value equal to upper boundary (3) Enter Phone No as valid numberic value	77-99999-7777777	Phone Number is accepted
TC_28		To verify the Phone number field with City Code more than upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value more than upper boundary (3) Enter Phone No as valid numberic value	77-100000-7777777	An error message should be displayed (Error ID - 20)
TC_29		To verify the Phone number field with Phone No less than lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-999999 77-77-0	An error message should be displayed (Error ID - 21)
TC_30		To verify the Phone number field with Phone No equal to the lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-1000000	Phone Number is accepted
TC_31		To verify the Phone number field with Phone No more than lower boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-1000001	Phone Number is accepted
TC_32		To verify the Phone number field with Phone No less than upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-999999998	Phone Number is accepted

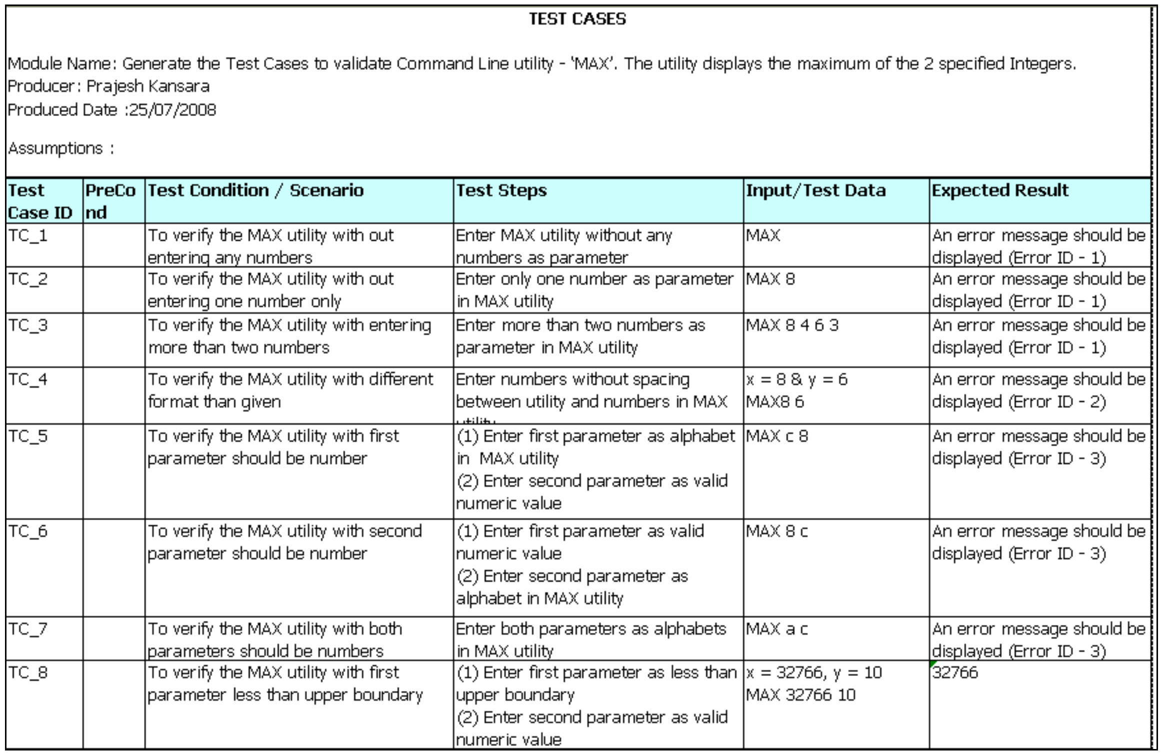


TC_33		To verify the Phone number field with Phone No equal to the upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-999999999	Phone Number is accepted
TC_34		To verify the Phone number field with Phone No more than upper boundary	(1) Enter Country Code as valid numeric value (2) Enter City Code as valid numeric value (3) Enter Phone No as valid numberic	77-77-1000000000	An error message should be displayed (Error ID - 22)
TC_35		To verify the valid Phone number field	Entering valid Phone number	10-10-1000000 11-10-1000000 998-10-1000000 999-10-1000000 10-11-1000000 10-99998-1000000 10-99999-1000000 10-10-1000001 10-10-999999998 10-10-999999999	Phone Number is accepted
TC_36		To verify the Country Code existance	Enter not existed Country Code in Phone number		An Error message should be displayed (Error ID - 23)
TC_37		To verify the City Code existance	Enter not existed City Code in Phone number		An Error message should be displayed (Error ID - 24)
TC_38		To verify that Emergancy Number	Enter Emergancy Number as Phone Number	100 101 102	An Error message should be displayed (Error ID - 25)

Error Message Table	
Error ID	**Error Description**
1	Please enter Phone Number
2	Please enter Country Code, it is blank in enter Phone number
3	Please enter City Code, it is blank in enter Phone number
4	Please enter Phone No, it is blank in enter Phone number
5	Please enter City Code and Phone No, it is blank in enter Phone number
6	Please enter Country Code and Phone No, it is blank in enter Phone number
7	Please enter Country Code and City Code, it is blank in enter Phone number
8	Please enter Country Code, City Code and Phone No; it is blank in enter Phone number
9	Please enter Phone Number is requier format
10	Country Code should be numeric
11	City Code should be numeric
12	Phone No should be numberic
13	Country Code and City Code should be numeric
14	Country Code and Phone No should be numeric
15	City Code and Phone No should be numeric
16	Country Code, City Code and Phone No should be numeric
17	Country Code is less than lower boundary
18	Country Code is more than upper boundary
19	City Code is less than lower boundary
20	City Code is more than upper boundary
21	Phone No is less than lower boundary
22	Phone No is more than upper boundary
23	Country Code don't exist in enter Phone number
24	City Code don't exist in enter Phone number
25	Emergancy number is enter as Phone number

04. Create test cases to validate the Command Line utility - 'MAX'

- Generate the Test Cases to validate Command Line utility - 'MAX'.
  - The utility displays the maximum of the 2 specified numbers. Please note down any assumptions that you make.
  - E.g. MAX 2 3



TEST CASES

Module Name: Generate the Test Cases to validate Command Line utility - 'MAX'. The utility displays the maximum of the 2 specified Integers.
Producer: Prajesh Kansara
Produced Date :25/07/2008

Assumptions :

Test Case ID	PreCond	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result
TC_1		To verify the MAX utility with out entering any numbers	Enter MAX utility without any numbers as parameter	MAX	An error message should be displayed (Error ID - 1)
TC_2		To verify the MAX utility with out entering one number only	Enter only one number as parameter in MAX utility	MAX 8	An error message should be displayed (Error ID - 1)
TC_3		To verify the MAX utility with entering more than two numbers	Enter more than two numbers as parameter in MAX utility	MAX 8 4 6 3	An error message should be displayed (Error ID - 1)
TC_4		To verify the MAX utility with different format than given	Enter numbers without spacing between utility and numbers in MAX	x = 8 & y = 6 MAX8 6	An error message should be displayed (Error ID - 2)
TC_5		To verify the MAX utility with first parameter should be number	(1) Enter first parameter as alphabet in MAX utility (2) Enter second parameter as valid numeric value	MAX c 8	An error message should be displayed (Error ID - 3)
TC_6		To verify the MAX utility with second parameter should be number	(1) Enter first parameter as valid numeric value (2) Enter second parameter as alphabet in MAX utility	MAX 8 c	An error message should be displayed (Error ID - 3)
TC_7		To verify the MAX utility with both parameters should be numbers	Enter both parameters as alphabets in MAX utility	MAX a c	An error message should be displayed (Error ID - 3)
TC_8		To verify the MAX utility with first parameter less than upper boundary	(1) Enter first parameter as less than upper boundary (2) Enter second parameter as valid numeric value	x = 32766, y = 10 MAX 32766 10	32766

TC_9		To verify the MAX utility with first parameter equal to the upper boundary	(1) Enter first parameter equal to the upper boundary (2) Enter second parameter as valid numeric value	x = 32767, y = 10 MAX 32767 10	32767
TC_10		To verify the MAX utility with first parameter more than upper boundary	(1) Enter first parameter more than upper boundary (2) Enter second parameter as valid numeric value	x = 32768, y = 10 MAX 32768 10	An error message should be displayed (Error ID - 4)
TC_11		To verify the MAX utility with second parameter less than upper boundary	(1) Enter first parameter as valid numeric value less than upper boundary (2) Enter second parameter less than	x = 10, y = 32766 MAX 10 32766	An error message should be displayed (Error ID - 6)
TC_12		To verify the MAX utility with second parameter equal to the upper boundary	(1) Enter first parameter as valid numeric value less than upper boundary (2) Enter second parameter equal to the upper boundary	x = 10, y = 32767 MAX 10 32767	An error message should be displayed (Error ID - 6)
TC_13		To verify the MAX utility with second parameter more than upper boundary	(1) Enter first parameter as valid numeric value less than upper boundary (2) Enter second parameter more	x = 10, y = 32768 MAX 10 32768	An error message should be displayed (Error ID - 4)
TC_14		To verify the MAX utility with first parameter less than lower boundary	(1) Enter first parameter as less than lower boundary (2) Enter second parameter as valid numeric value	x = -32767, y = 10 MAX -32767 10	An error message should be displayed (Error ID - 6)
TC_15		To verify the MAX utility with first parameter equal to the lower boundary	(1) Enter first parameter equal to the lower boundary (2) Enter second parameter as valid numeric value	x = -32768, y = 10 MAX 32768 10	An error message should be displayed (Error ID - 6)
TC_16		To verify the MAX utility with first parameter more than lower boundary	(1) Enter first parameter more than lower boundary (2) Enter second parameter as valid numeric value	x = -32769, y = 10 MAX -32769 10	An error message should be displayed (Error ID - 4)
TC_17		To verify the MAX utility with second parameter less than lower boundary	(1) Enter first parameter as valid numeric value less than lower boundary (2) Enter second parameter less than	x = 10, y = -32767 MAX 10 -32767	An error message should be displayed (Error ID - 5)
TC_18		To verify the MAX utility with second parameter equal to the lower boundary	(1) Enter first parameter as valid numeric value less than lower boundary (2) Enter second parameter equal to	x = 10, y = -32768 MAX 10 -32768	An error message should be displayed (Error ID - 5)
TC_19		To verify the MAX utility with second parameter more than lower boundary	(1) Enter first parameter as valid numeric value less than lower boundary (2) Enter second parameter more	x = 10, y = -32769 MAX 10 -32769	An error message should be displayed (Error ID - 4)
TC_20		To verify the MAX utility with both parameters less than upper boundary	Enter both parameters less than upper boundary	x = 32766, y = 32766 MAX 32766 32766	An error message should be displayed (Error ID - 7)
TC_21		To verify the MAX utility with both parameters equal to the upper boundary	Enter both parameters equal to the upper boundary	x = 32767, y = 32767 MAX 32767 32767	An error message should be displayed (Error ID - 7)
TC_22		To verify the MAX utility with both parameters more than upper boundary	Enter both parameter more than upper boundary	x = 32768, y = 32768 MAX 32768 32768	An error message should be displayed (Error ID - 4)
TC_23		To verify the MAX utility with both parameters less than lower boundary	Enter both parameters less than lower boundary	x = -32767, y = -32767 MAX -32767 -32767	An error message should be displayed (Error ID - 7)
TC_24		To verify the MAX utility with both parameters equal to the lower boundary	Enter both parameters equal to the lower boundary	x = -32768, y = -32768 MAX -32768 -32768	An error message should be displayed (Error ID - 7)
TC_25		To verify the MAX utility with both parameters more than lower boundary	Enter both parameter more than lower boundary	x = -32769, y = -32769 MAX -32769 -32769	An error message should be displayed (Error ID - 4)
TC_26		To verify the MAX utility with first parameter is grater than second	Enter first valid parameter grater than second valid parameter	x = 10, y = 5 MAX 10 5	An error message should be displayed (Error ID - 5)
TC_27		To verify the MAX utility with first parameter is less than second	Enter first valid parameter less than second valid parameter	x = 5, y = 10 MAX 5 10	An error message should be displayed (Error ID - 6)
TC_28		To verify the MAX utility with two same parameters	Enter first and second parameters with same valid value	x = 5, y = 5 MAX 5 5	An error message should be displayed (Error ID - 7)

Error Message Table	
Error ID	**Error Description**
1	Please execute MAX command with required data
2	Please execute MAX command with prescribed format
3	Please enter numerical values as parameter
4	Please enter numbers in valid range
5	First parameter is Maximum
6	Second parameter is Maximum
7	Both parameters are same

05. Create test cases to validate the Command Line utility - 'SORT'

Testing the 'SORT' utility

❖ This is a command line SORT Utility. User instructions for this utility are as follows:

1. Execute the command at command prompt as follows:
   SortUtility <inputfile>
   E.g SortUtility EmpNames
2. Utility creates the output file as **<inputfile>.sorted.**
3. The input file contains one employee name on each line.
4. Valid Employee name can be a maximum of 50 characters.
5. Only special characters allowed are apostrophes and spaces.

❖ Develop test cases to test the above utility.

TEST CASES

Module Name: Testing the 'SORT' utility
Producer: Prajesh Kansara
Produced Date :25/07/2008
Assumptions : As per specification SortUtility command work perfectly and take any type of ASCII file name (with/without any extension) as parameter

Test Case ID	PreCond	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result
TC_1		To verify the SortUtility with out entering any file name	Enter SortUtility without any file name as parameter	SortUtility	Error message should be display **"Please execute SortUtility command with required parameters"**
TC_2		To verify the SortUtility with entering more than one file names	Enter more than one file name as parameter in SortUtility	SortUtility ABC.txt XYZ.txt	Error message should be display **"Please execute SortUtility command with required parameters"**
TC_3		To verify the SortUtility with different format	Enter file name in different format than specified format	SortUtilityABC.txt SortUtility#ABC.txt	Error message should be display **"No such command found"**
TC_4		To verify that valid command is given	Enter file name with some other command name	Sort ABC.txt	Error message should be display **"No such command found"**
TC_5		To verify that SortUtility command should be executed with invalid file name	Enter invalid file name (containing \, /, :, /, ", <, >, | in file name) as parameter in SortUtility command on the command prompt in valid format	SortUtility <N|A*.txt	Error message should be display **"Invalid file name (File name can't contain - \, / , :, ", ?, <. >)"**

TC_6		To verify the output of SortUtility command with valid file name	Enter valid file name as parameter in SortUtility command on command prompt in valid format	SortUtility ABC.txt Data in ABC.txt Prajesh Apu Raj	(1) SortUtility command accept file name (ABC.txt) and executing command successfully. (2) Generate sorted output file successfully with ABC.sorted name (3) ABC.sorted file should contain following data Apu Prajesh Raj
TC_7		To verify the output of SortUtility command with valid empty file	Enter valid empty file name as parameter in SortUtility command on command prompt in valid format	SortUtility Empty.txt **Data in Empty.txt**	(1) SortUtility command accept file name (Empty.txt) and executing command successfully. (2) Generate sorted output file successfully with Empty.sorted name (3) Empty.sorted file should be
TC_8		To verify the output of SortUtility command with valid file name containing more than 50 character in one line	Enter valid file name as parameter in SortUtility command, where file contain more than 50 character in one line	SortUtility LargeRow.txt **Data in LargeRow.txt** ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFGHIJKLMNOPQRSTUVWXYZ	(1) SortUtility command accept file name (LargeRow.txt) and execution terminated. (2) Error message should be display **"File contain row(s) with employee name more than 50 characters"** (3) Output file with name LargeRow.sorted should not be generated.

TC_9		To verify the output of SortUtility command with valid file name that do not containing only characters and special characters like apostrophes and spaces as employee name in each row	Enter valid file name as parameter in SortUtility command, where file contain special symbols and numerical numbers as employee name in row(s)	SortUtility ABC.txt Data in Data.txt 12Prajesh$$ Apu1 #Rah# Raj	(1) SortUtility command accept file name (Data.txt) and execution terminated. (2) Error message should be display **"File contain row(s) with invalid employee name"** (3) Output file with name Data.sorted should not be generated.
TC_10		To verify the output of SortUtilify command with valid file name that containing more than one similar special character together in employee name	Enter valid file name as parameter in SortUtility command, where file contain more than one special character together in employee name in row(s)	SortUtility ABC.txt Data in Input.txt Prajesh Kansara De''Costa Raj	(1) SortUtility command accept file name (Input.txt) and execution terminated. (2) Error message should be display **"File contain row(s) with more than one special characters together in employee name"** (3) Output file with name Input.sorted should not be
TC_11		To verify the output of SortUtilify command with valid file name that containing more than one similar employee name	Enter valid file name as parameter in SortUtility command, where file contain more than one similar employee name in row(s)	SortUtility ABC.txt Data in Input.txt Prajesh Kansara Raj Prajesh Kansara	(1) SortUtility command accept file name (Input.txt) and executing command successfully. (2) Generate sorted output file successfully with Input.sorted name (3) ABC.sorted file should contain following data Prajesh Kansara Prajesh Kansara Raj

TC_12		To verify the output of SortUtilify command with valid file name with out read access control	Enter valid file name as parameter in SortUtility command, where file with out read access control	SortUtility ABC.txt Data in Input.txt Prajesh Kansara Raj Prajesh Kansara	(1) SortUtility command accept file name (Input.txt) and execution terminated. (2) Error message should be display **"You don't have read access of the source file contain employee name"** (3) Output file with name Input.sorted should not be generated.
TC_13		To verify the output of SortUtilify command with valid file name which don't exist	Enter valid file name as parameter in SortUtility command, where file do not exist	SortUtility Temp.txt	(1) SortUtility command accept file name (Temp.txt) and execution terminated. (2) Error message should be display **"File do not exists"** (3) Output file with name Temp.sorted should not be generated.
TC_14		To verify the output of SortUtilify command with valid file name and output file all ready exists in same directory	Enter valid file name as parameter in SortUtility command, where output file all ready exists in the same directory	SortUtility ABC.txt Data in ABC.txt Prajesh Apu Raj	(1) SortUtility command accept file name (ABC.txt) . (2) Output file ABC.sorted exists than error message should be display "Sorted file all ready exists. Would you like to over write the file" (3) If user wish to over write the existed output file ABC.sorted then SortUtility command execute and over write the content of ABC.sorted file sorted data according to source file ABC.txt. Data in ABC.sorted Apu Prajesh Raj (4) If user do not wish to over write the output file with name ABC.sorted then SortUtility command will terminated and contents of ABC.sorted will not change.
TC_15		To verify the output of SortUtilify command with valid file name and user don't have rights to create new output file	Enter valid file name as parameter in SortUtility command, where user don't have rights to create new output file	SortUtility ABC.txt Data in ABC.txt Prajesh Apu Raj	(1) SortUtility command accept file name (ABC.txt) and execution terminated. (2) Error message should be display **"Do not have rights to generate sorted file".** (3) Output file with name ABC.sorted should not be generated.
TC_16		To verify the output of SortUtility command with valie invalide file name according to DOS	Enter invalide file name accprdomg to DOS as parameter in SortUtility command system	SortUtility ABCDEFGHI.txt SortUtility ABC.dump	Error message should be display "Invalide file name"

====**Topic - 7**=== XXX ===**TEST CASES**===== XXX =====================