IT8076 **SOFTWARE TESTING** **L T P C**

3 0 0 3

OBJECTIVES:

To learn the criteria for test cases.

To learn the design of test cases.

To understand test management and test automation techniques.

To apply test metrics and measurements.

UNIT I INTRODUCTION 9

Testing as an Engineering Activity – Testing as a Process – Testing Maturity Model- Testing axioms – Basic definitions – Software Testing Principles – The Tester's Role in a Software Development Organization – Origins of Defects – Cost of defects – Defect Classes – The Defect Repository and Test Design –Defect Examples- Developer/Tester Support of Developing a Defect Repository.

UNIT II TEST CASE DESIGN STRATEGIES 9

Test case Design Strategies – Using Black Box Approach to Test Case Design – Boundary Value Analysis – Equivalence Class Partitioning – State based testing – Cause-effect graphing – Compatibility testing – user documentation testing – domain testing - Random Testing – Requirements based testing – Using White Box Approach to Test design – Test Adequacy Criteria – static testing vs. structural testing – code functional testing – Coverage and Control Flow Graphs – Covering Code Logic – Paths – code complexity testing – Additional White box testing approaches- Evaluating Test Adequacy Criteria.

UNIT III LEVELS OF TESTING 9

The need for Levels of Testing – Unit Test – Unit Test Planning – Designing the Unit Tests – The Test Harness – Running the Unit tests and Recording results – Integration tests – Designing Integration Tests – Integration Test Planning – Scenario testing – Defect bash elimination System Testing – Acceptance testing – Performance testing – Regression Testing – Internationalization testing – Ad-hoc testing – Alpha, Beta Tests – Testing OO systems – Usability and Accessibility testing – Configuration testing –Compatibility testing – Testing the documentation – Website testing.

UNIT IV TEST MANAGEMENT 9

People and organizational issues in testing – Organization structures for testing teams – testing services – Test Planning – Test Plan Components – Test Plan Attachments – Locating Test Items – test management – test process – Reporting Test Results – Introducing the test specialist – Skills needed by a test specialist – Building a Testing Group- The Structure of Testing Group- .The Technical Training Program.

Arunai Engineering College

UNIT V TEST AUTOMATION 9

Software test automation – skills needed for automation – scope of automation – design and architecture for automation – requirements for a test tool – challenges in automation – Test metrics and measurements – project, progress and productivity metrics

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course the students will be able to:

Design test cases suitable for a software development for different domains.

Identify suitable tests to be carried out.

Prepare test planning based on the document.

Document test plans and test cases designed

Use automatic testing tools

Develop and validate a test plan

TEXT BOOKS:

1. Srinivasan Desikan and Gopalaswamy Ramesh, ―Software Testing – Principles and Practices‖, Pearson Education, 2006.

2. Ron Patton, ―Software Testing‖, Second Edition, Sams Publishing, Pearson Education, 2007. AU Library.com

REFERENCES:

1. Ilene Burnstein, ―Practical Software Testing‖, Springer International Edition, 2003.

2. Edward Kit,‖ Software Testing in the Real World – Improving the Process‖, Pearson Education, 1995.

3. Boris Beizer,‖ Software Testing Techniques‖ – 2nd Edition, Van Nostrand Reinhold, New York, 1990.

4. Aditya P. Mathur, ―Foundations of Software Testing _ Fundamental Algorithms and Techniques‖, Dorling Kindersley (India) Pvt. Ltd., Pearson Education, 2008.

Arunai Engineering College

ARUNAI ENGINEERING COLLEGE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
IT8076 -SOFTWARE TESTING (2017 Regulation)

UNIT I INTRODUCTION
Testing as an Engineering Activity – Testing as a Process – Testing Maturity Model- Testing axioms –Basic definitions – Software Testing Principles – The Tester's Role in a Software Development Organization – Origins of Defects – Cost of defects – Defect Classes – The Defect Repository and Test Design –Defect Examples- Developer/Tester Support of Developing a Defect Repository.

Software Testing : Introduction

IEEE Definition :

A process of analyzing a s/w item to detect the difference between existing and required conditions (error /defect / bugs) and to evaluate the features of the s/w item .

Definition 1:

Process of evaluating a system or its components with the intent to find whether it satisfies the specified requirements or not.

Definition 2:

an investigation conducted to provide stakeholders with information about the quality of the product or service.

Who is involved in Testing ?

Test Specialist

S/w Developer

Project leader Manager

End User

When to start the testing process?

Every Phase of SDLC

When to end the testing process?

When Testing Deadline comes

Completion of test case execution

Bug rate falls below certain level & no high priority bugs are identified

Management decision

Example : S/W Testing Automation Tools

Telerik Test Studio , Selenium , TestComplete , HP Quick Test Professional , Silk Test, Win Runner ,Load Runner

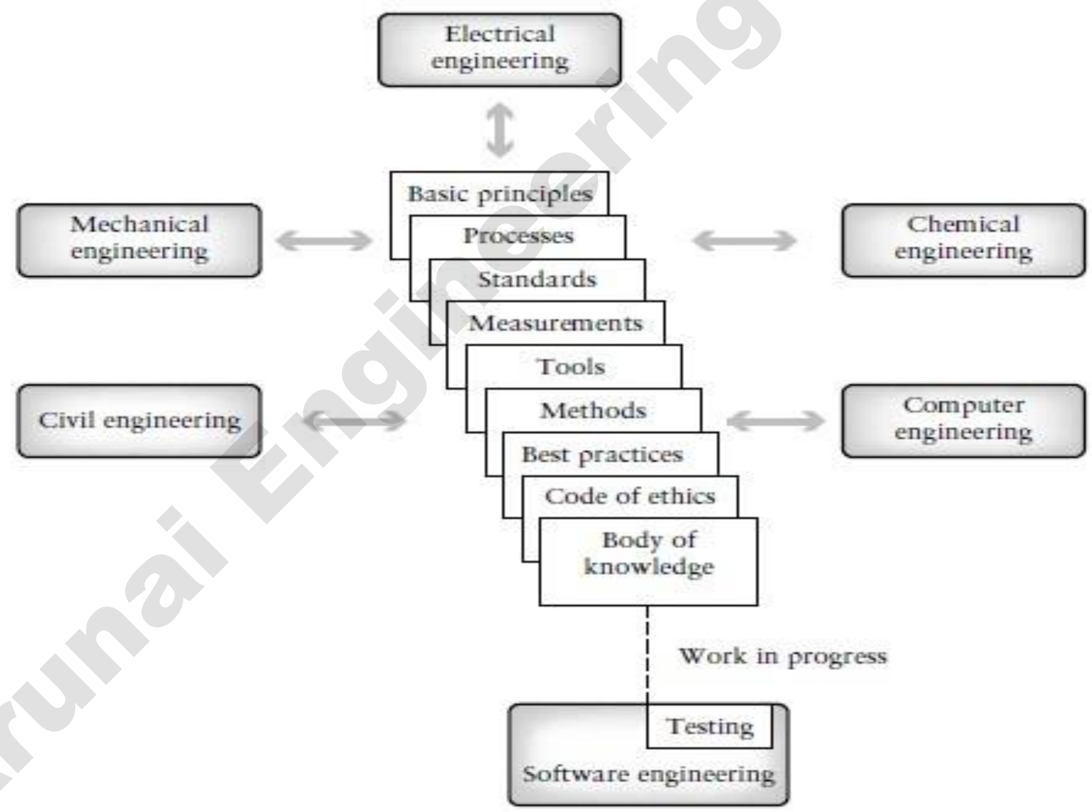
Arunai Engineering College

Testing as an Engineering Activity:-

Poor quality of software that can cause loss of life or property is no longer acceptable to society .Failures can result in catastrophic losses. Conditions demand software development staffs with interest and training in the areas of software products and process quality. Highly qualified staffs ensures that software products are built on time, within budget and are of the highest quality with respect to attributes such as reliability, correctness, usability and ability to meet all user requirements.

The demand for high quality software's and the need for well educated software professionals there is a movement to change the way software is developed and maintained.

The profession of software engineering is slowly emerging as a formal engineering discipline .The movement towards this new profession is the focus of the entire November /December 1999 issues of *IEEE Software*.



The education and training of engineers in each engineering discipline is based on the teaching of related scientific principles, engineering processes, standards, methods, tools, measurements. The goal and task force team is to define a body of knowledge that covers the software engineering discipline, to discuss the nature of education for this new profession and to define a code of ethics for the software engineers

Arunai Engineering College

Using Engineering Approach to software development implies that:-

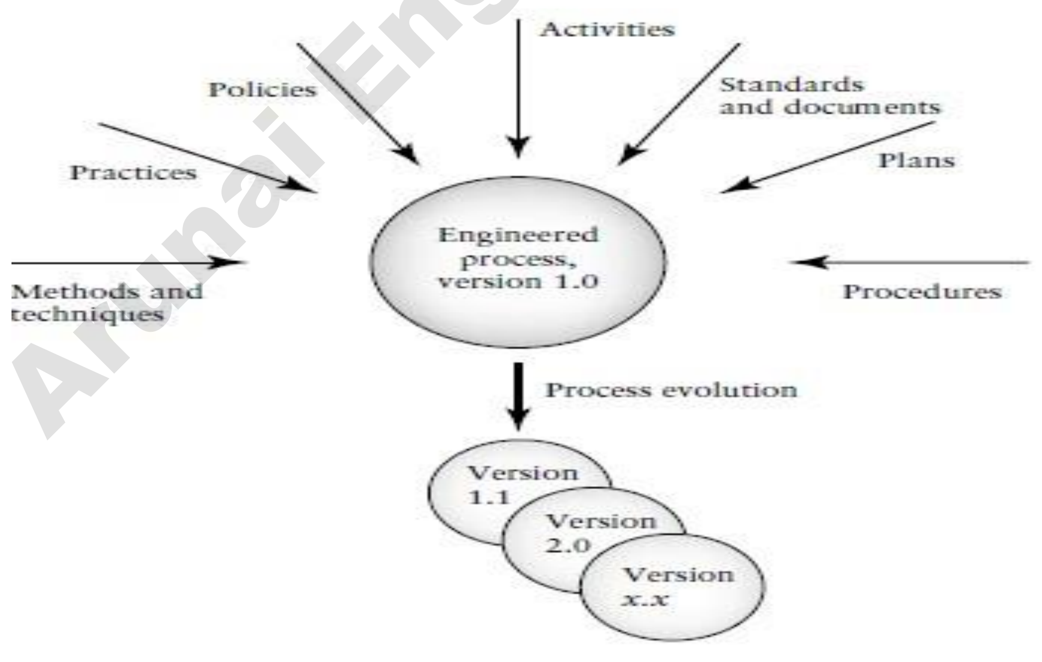
- The development process is well understood
- project are planned
- life cycle models are defined and adhered to
- standards are in place for products and process
- measurements are employed to evaluate products and process quality
- Components are reused.
- Validation and verification processes play a key role in quality determination

Engineers have proper education, training and certification

A test specialist is one whose education is based on the principles , practices and processes that constitute the software engineering discipline, and whose specific focus is on one area of that discipline – software testing . A test specialist who is trained as an engineer should have knowledge of test related principles, processes, measurements, standards, plans, tools and methods, and should learn how to apply them to the testing tasks to be performed.

Role of Process in Software Quality:-

Process, in the software engineering domain , is the set of methods, practices, standards , documents , activities , policies , and procedures that software engineers use to develop and maintain a software system and its associated artifacts , such as project and test plans, design documents code and manuals



The software development process like most engineering artifacts must be engineered . Software process improvement models that have had wide acceptance in

Arunai Engineering College

industry are high level models, in the sense that they focus on the software process as a whole and do not offer adequate support to evaluate and import specific software development sub processes such as design and testing.

Testing as a Process:-

Process has been described as a series of phases, procedures, and steps that result in the production of a software product. Embedded within this are several processes.

Validation is the process of evaluating a software system or components during or at the end of, the development cycle in order to determine whether it satisfies specified requirements

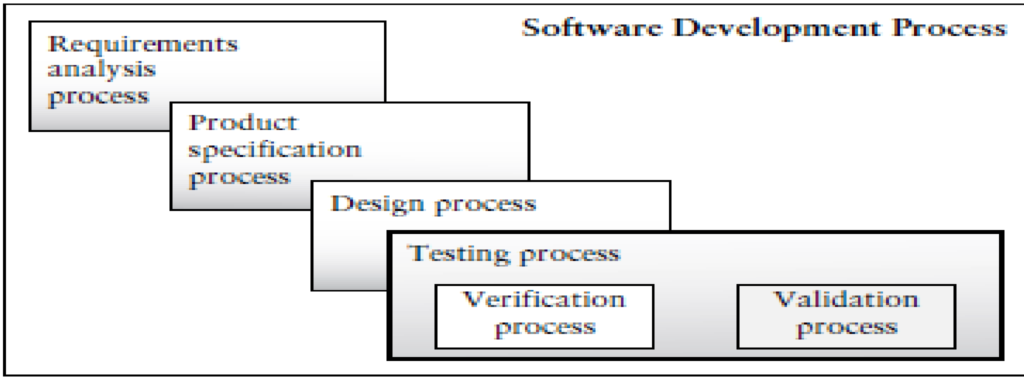
Validation is usually associated with traditional execution based testing that is exercising the code with test cases

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Verification is usually associated with activities such as inspections and reviews of software deliverables.

Difference Between Verification & Validation

Verification	Validation
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. Verification is to check whether the software conforms to specifications.	5. Validation is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after verification.

Arunai Engineering College



Testing is generally described as a process as a group of procedures carried out to evaluate some aspects of a piece of software.

Testing can be described as a process used for revealing defects in software, and establishing that the software has attained a specified degree of quality with respect to selected attributes.

Testing and Debugging are two different activities.

Debugging process begins after localization has been carried out and the tester has noted that the software is nit behaving as specified

Debugging or Fault Localization is the process of

(1) Locating the fault or defect

(2) Repairing the code , and

(3) Retesting the code

Testing as a process has economic, technical and managerial aspects.

- **Economic aspects** are related to the reality that resources and time are available to the testing group on a limited basis
- **Technical Aspect** of testing are relate to the techniques, methods, measurements, and tools used to insure that the software under test is a defect free and reliable as possible for the conditions and constraints under which it must operate
- **Managerial Aspect –** Minimally that means that an organizational policy for testing must be defined and documented. Testing must be planned, testers should be trained, the process should have associated quantifiable goals that can be measured and monitored.

Testing Maturity Model - Introduction

To know about Testing , one must find answer for following queries

- Where do we begin to learn more about testing ?
- What areas of testing are important?
- Which topics need to be addressed first?

TMM gives answer for all the questions listed.

Testing Maturity Model - Definition

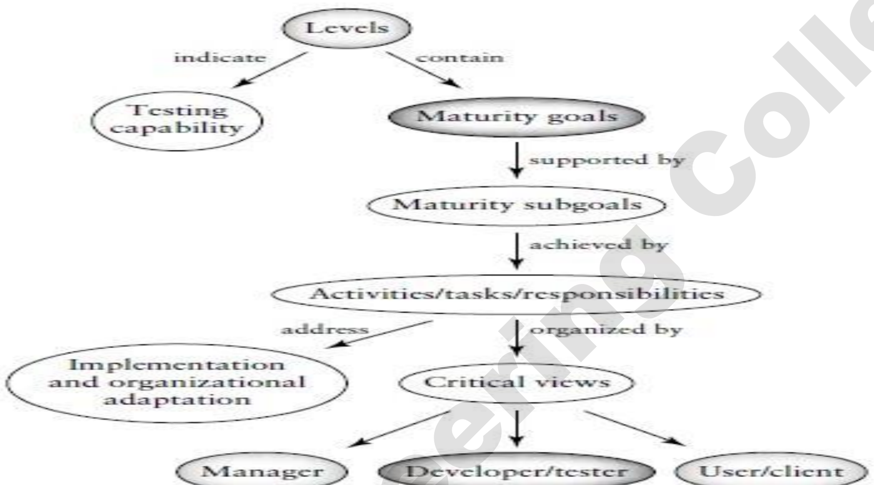
TMM is a learning tools ,or framework to learn about testing. It introduces both the technical and managerial aspects of testing. It evolve testing process both in the

Arunai Engineering College

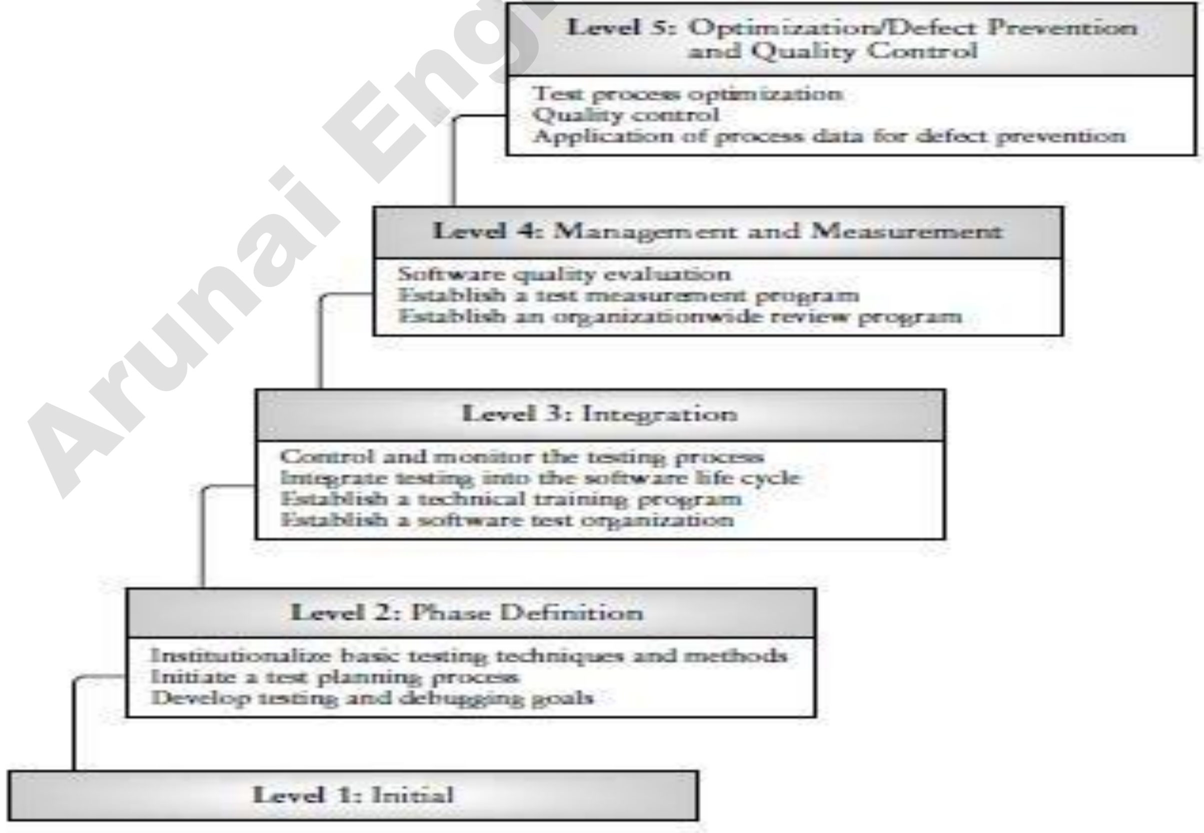
personal and organizational levels. It follows staged architecture for process improvement models.
It has five levels that prescribe a maturity hierarchy and an evolutionary path to test process improvement. Each level has (Except Level 1)

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

Internal Structure of TMM maturity model



<u>**Testing Maturity Model - *5-level structure***</u>



Arunai Engineering College

Level 1—Initial: (No maturity goals)

testing is a chaotic process; it is ill-defined
Not distinguished from debugging.
The objective of testing is to show the software works
Software products are often released without quality assurance.
lack of resources, tools and properly trained staff.

Level 2—Phase Definition:

Goal 1: Develop testing and debugging goals;
Goal 2: Initiate a testing planning process;
Goal 3: Institutionalize basic testing techniques and methods
testing is separated from debugging and is defined as a phase that follows coding.
It is a planned activity; however, test planning at level 2 may occur after coding for reasons related to the immaturity of the testing process.
use of black box and white box testing strategies, and a validation cross-reference matrix
Testing is multileveled - unit, integration, system, and acceptance levels.

Level 3—Integration

Goal 1: Establish a software test organization;
Goal 2: Establish a technical training program;
Goal 3: Integrate testing into the software life cycle;
Goal 4: Control and monitor testing
testing is integrated into the entire software life cycle
There is a test organization, and testing is recognized as a professional activity.
There is a technical training organization with a testing focus
Testing is monitored to ensure it is going according to plan and actions can be taken if deviations occur

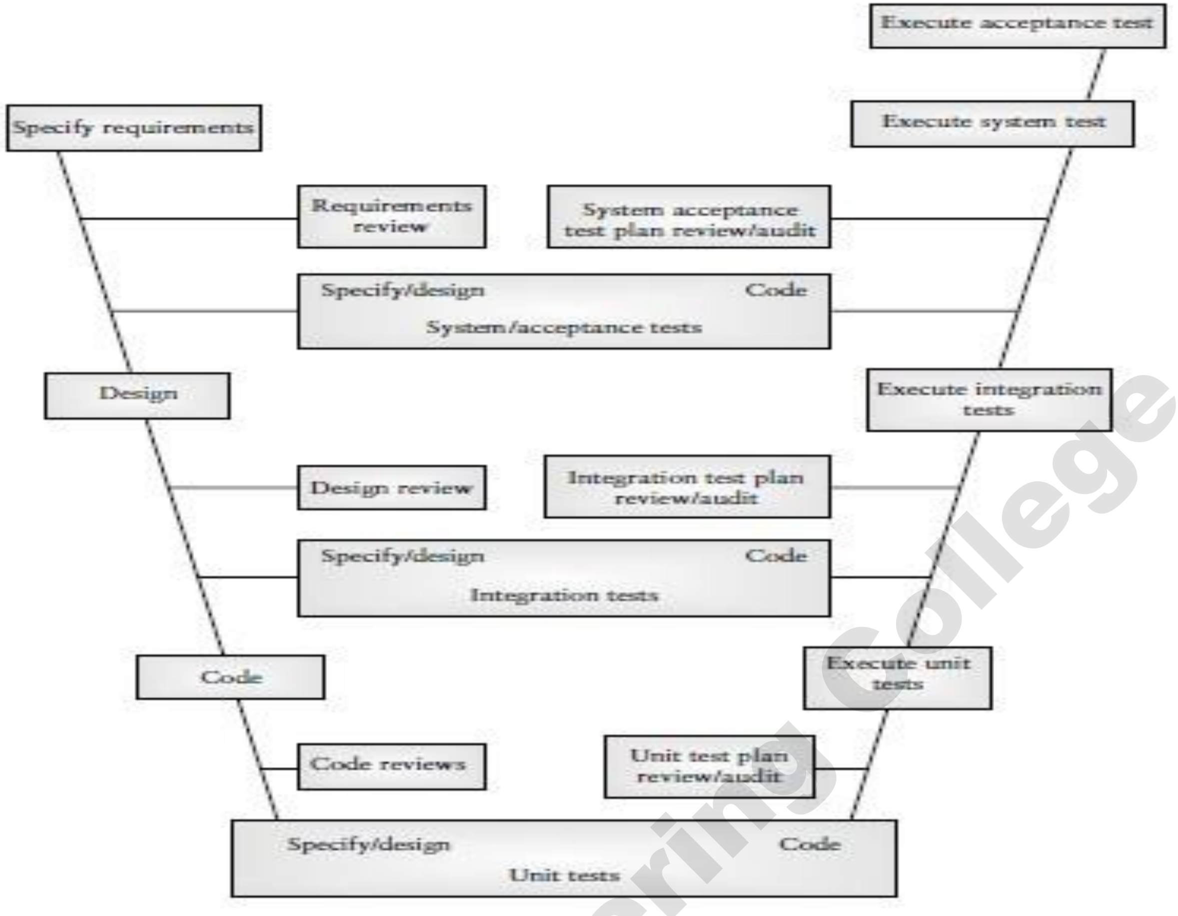
Level 4—Management and Measurement

Goal 1: Establish an organization wide review program;
Goal 2: Establish a test measurement program;
Goal 3: Software quality evaluation
process that is measured and quantified. Reviews at all phases of the development process are now recognized as testing/quality control activities.
Software products are tested for quality attributes such as reliability, usability, and maintainability.
Test cases from all projects are collected and recorded in a test case database for the purpose of test case reuse and regression testing. Defects are logged and given a severity level.
Some of the deficiencies occurring in the test process are due to the lack of a defect prevention philosophy. An extension of the V-model as shown in Figure can be used to support the implementation of this goal

Level 5—Optimization/Defect Prevention/Quality Control

Goal 1: Defect prevention;
Goal 2: Quality control;
Goal 3: Test process optimization
the testing process is now said to be defined and managed; its cost and effectiveness can be monitored. Defect prevention and quality control are practiced. Automated tools totally support the running and rerunning of test cases

Arunai Engineering College



Extension of the V-model

Testing Axioms

Axioms: "rules of the road" or the "facts of life" for software testing and software development.

1. **It's Impossible to Test a Program Completely**
2. **Software Testing Is a Risk-Based Exercise**
3. **Testing Can't Show That Bugs Don't Exist**
4. **The More Bugs You Find, the More Bugs There Are**
5. **The Pesticide Paradox**
6. **Not All the Bugs You Find Will Be Fixed**
7. **When a Bug's a Bug Is Difficult to Say**
8. **Product Specifications Are Never Final**
9. **Software Testers Aren't the Most Popular Members of a Project Team**
10. **Software Testing Is a Disciplined Technical Profession**

1. It's Impossible to Test a Program Completely

due to four key reasons:

• The number of possible inputs is very large.

Arunai Engineering College

• The number of possible outputs is very large.
• The number of paths through the software is very large.
• The software specification is subjective

Ex: Microsoft Windows Calculator

Assume that you are assigned to test the Windows Calculator. You decide to start with addition. You try 1+0=. You get an answer of 1. That’s correct. Then you try 1+1=. You get 2. How far do you go? The calculator accepts a 32-digit number, so you must try all the possibilities up to

1+99999999999999999999999999999999=

Once you complete that series, you can move on to 2+0=, 2+1=, 2+2=, and so on. Eventually you’ll get to

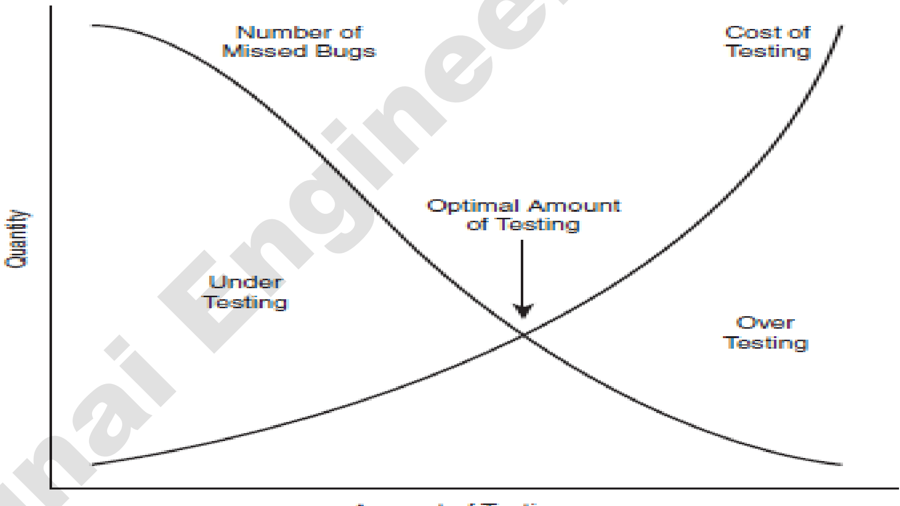
99999999999999999999999999999999+99999999999999999999999999999999=

Next you should try all the decimal values: 1.0+0.1, 1.0+0.2, and so on.

it’s impossible to completely test a program, even software as simple as a calculator. If you decide to eliminate any of the test conditions because you feel they’re redundant or unnecessary, or just to save time, you’ve decided not to test the program completely.

2. Software Testing Is a Risk-Based Exercise

One key concept that software testers need to learn is how to reduce the huge domain of possible tests into a manageable set, and how to make wise risk-based decisions on what’s important to test and what’s not.



Above graph shows the relationship between the amount of testing performed and the number of bugs found. If you attempt to test everything, the costs go up dramatically and the number of missed bugs declines to the point that it’s no longer cost effective to continue.

If you cut the testing short or make poor decisions of what to test, the costs are low but you’ll miss a lot of bugs. The goal is to hit that optimal amount of testing so that you don’t test too much or too little.

3. Testing Can’t Show That Bugs Don’t Exist

You’re an exterminator charged with examining a house for bugs. You inspect the house and find evidence of bugs

Arunai Engineering College

House1 :

Findings :—maybe live bugs, dead bugs, or nests.
Conclusion :- You can safely say that the house has bugs.

House2 :

Findings :- no evidence of bugs. e no signs of an infestation.
Maybe you find a few dead bugs or old nests but you see nothing that tells you that live bugs exist.
Conclusion : your search you didn't find any live bugs. Unless you completely dismantled the house down to the foundation, you can't be sure that you didn't simply just miss them.

Software testing works exactly as the exterminator does. It can show that bugs exist, but it can't show that bugs don't exist. You can perform your tests, find and report bugs, but at no point can you guarantee that there are no longer any bugs to find.

4. The More Bugs You Find, the More Bugs There Are

Reasons

Programmers have bad days. Like all of us, programmers can have off days. Code written one day may be perfect; code written another may be sloppy.
Programmers often make the same mistake. Everyone has habits. A programmer who is prone to a certain error will often repeat it.
Some bugs are really just the tip of the iceberg. Very often the software's design or architecture has a fundamental problem. A tester will find several bugs that at first may seem unrelated but eventually are discovered to have one primary serious cause.

5. The Pesticide Paradox

The test process repeats each time around the loop. With each iteration, the software testers receive the software for testing and run their tests. Eventually, after several passes, all the bugs that those tests would find are exposed. Continuing to run them won't reveal anything new.
To overcome the pesticide paradox, software testers must continually write new and different tests to exercise different parts of the program and find more bugs.

6. Not All the Bugs You Find Will Be Fixed

reasons why you might choose not to fix a bug:

• There's not enough time. In every project there are always too many software features, too few people to code and test them, and not enough room left in the schedule to finish. If you're working on a tax preparation program, April 15 isn't going to move—you must have your software ready in time.
• It's really not a bug. Maybe you've heard the phrase, "It's not a bug, it's a feature!" It's not uncommon for misunderstandings, test errors, or spec changes to result in would-be bugs being dismissed as features.

Arunai Engineering College

• It's too risky to fix. You might make a bug fix that causes other bugs to appear. Under the pressure to release a product under a tight schedule, it might be too risky to change the software. It may be better to leave in the known bug to avoid the risk of creating new, unknown ones.
• It's just not worth it. This may sound harsh, but it's reality. Bugs that would occur infrequently or bugs that appear in little-used features may be dismissed.

7. When a Bug's a Bug Is Difficult to Say

rules to define a bug

1. The software doesn't do something that the product specification says it should do.
2. The software does something that the product specification says it shouldn't do.
3. The software does something that the product specification doesn't mention.
4. The software doesn't do something that the product specification doesn't mention but should.
5. The software is difficult to understand, hard to use, slow, or—in the software tester's eyes—will be viewed by the end user as just plain not right.

8. Product Specifications Are Never Final

You're halfway through the planned two year development cycle, and your main competitor releases a product very similar to yours but with several desirable features that your product doesn't have.

- Do you continue with your spec as is and release an inferior product in another year?
- Or, does your team regroup, rethink the product's features, rewrite the product spec, and work on a revised product?

9. Software Testers Aren't the Most Popular Members of a Project Team

The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

- Find bugs early.
- Temper your enthusiasm
- Don't always report bad news

10. Software Testing Is a Disciplined Technical Profession

- If software testers were used, they were frequently untrained and brought into the project late to do some "ad-hoc banging on the code to see what they might find." Times have changed.
- The software industry has progressed to the point where professional software testers are mandatory. It's now too costly to build bad software.

Basic Definitions:-

1. Error :-

A Error is a mistake, misconception, or misunderstanding on the part of a software developers. Developers we include software Engineers, programming analysts and testers. It is the terminology of the developer. For Eg, a developer may understand a design notation, or a programmer might type a variable name incorrectly.

2. Faults / Defect :-

A fault(Defects) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly , and not according to

Arunai Engineering College

its specification , Faults or defects are sometimes called as "**bugs".** It is the terminology of the tester.

3. Failures

A failure is the inability of a software or component to perform its required functions within specified performance requirements. It is the terminology of the customer. Error leads to Defect , Defect leads to Failure.

A Fault in the code does not always produce a failure. In fact, faulty software may operate over a long period of time without exhibiting any in correct behavior. When the proper conditions occur the fault will manifest itself as a failure.

1.. The input to the software must cause the faulty statements to be executed
2. The faulty statements must produce different results than the correct statements. This event produces an incorrect internal state for the software.
3. The incorrect internal state must propagate to the output, so that the result of the fault is observable

4. **Test Cases:-**

Detecting defects in a piece of software is for the tester to select a set of input data and then execute the software with the input data under a particular set of conditions. To check the software is success or failed the tester need to know the output of the software, input of the software and soon

A test case in a practical sense is a test related item which contains the following information:-

❖ *A set of test inputs :-* These are data items received from an external source by the code under test. The external source can be hardware , software or human.
❖ *Execution Condition:-* These are conditions required for running the test, for example , a certain state of database, or a configuration of hardware devices
❖ *Expected Outputs:-* These are the specified results to be produced by the code under test

Ex: biggest of 3 Numbers

Test Case Id	Test I/p	Expected O/P	Actual O/p	Result :Pass/Fail
TC1	A=10 B=20 C=50	C IS BIG		
TC2	A=110 B=20 C=50	A IS BIG		
TC3	A=10 B=120 C=50	B IS BIG		

5. **Test :-**

A test is a group of related test cases, or a group of related test cases and test procedures(steps needed to carry out a test)

A group of related tests is sometimes referred to as a test set. A group of related tests that are associated with the database, and are usually run together is sometimes referred to as test suite

Arunai Engineering College

6. **Test Oracle:-**

A test oracle is a document, or piece of software that allows testers to determine whether a test has been passed or failed.

A program, or a document that produces or specifies the expected outcome of a test, can serve as an oracle. Example include a specification, design documents, set of requirements.

Other sources are regression test suites. The suite usually contains components with correct results for previous versions of software.

7. **Test Bed:-**

A test bed is an environment that contains all the hardware and software needed to test a software components or a software system.

This include testing environment, eg :- simulators, emulators memory checkers, hardware probes, software tools etc

8. **Software Quality :-**

Two concise definition for quality are found in the IEEE standards Glossary of Software Engg Terminology

a. Quality relates to the degree to which a system, system component, or process meets specified requirements

b. Quality relates to the degree to which a system, system components , or process meets customer or user needs, or expectations

Software artifacts we can measure the degree to which they posses a given quality attribute with quality metrics.

9. **Metrics:-**

A metrics is a quantitative measure of the degree to which a system, system component or process possesses a given attribute. There are product and process metrics. A very commonly used example software products metrics is software size, usually measured in Lines Of Code(LOC)

10. **Quality Metrics:-**

Is a quantitative measurement of the degree to which an Item possesses a given quality attribute. Quality attributes with brief explanation are the following :-

i. **Correctness:-** the degree to which the system performs its intended function
ii. **Reliability :-** the degree to which the software is expected to perform its required function under stated conditions for a stated period of time
iii. **Usability :-** related to the degree of effort needed to learn , operate, prepare input, and interpret output of the software.
iv. **Integrity:-** Relates to the system's ability to withstand both intentional and accidental attacks
v. **Portability:-** Relates to the ability of the software to be transferred from one environment to another.
vi. **Maintainability:-** the effort needed to make change in the software
vii. **Interoperability: -** the effort needed to link or couple one system to another.

Testability attribute is of more interest to developers/testers than to clients. It can be expressed in the following two ways:-

Arunai Engineering College

a) The amount of effort needed to test the software to ensure it performs according to specified requirements (relates the number of test cases needed)

b) The ability of the software to reveal defects under testing conditions (some s/w is designed in such a way that defects are well hidden during ordinary testing conditions)

11. **Software Quality Assurance Group:-**

SQA group in an organization has ties to quality issues. The Software Quality Assurance group SQA is a team of people with necessary training and skills to ensure that all necessary actions are taken during the development process so that the resulting software conforms to established technical requirements

12. **Reviews:-**

Dynamic execution based testing techniques that can be used to detect defects and evaluate software quality, review are types of static testing techniques that can be used to evaluate the quality of the software artifacts such as requirement documents , a test plan, a design document ,a code component.

A review is a group meeting whose purpose is to evaluate a software artifact or a set of software artifacts.

Software Testing Principles:-

Principle: Fundamental to the objective of testing, namely , to provide quality products to customers. The Fundamental principles of testing are as follows

1. **The Goal of testing is to find defects before customers find them out.**
   Ex: The Incomplete Car
2. **Exhaustive testing is not possible; program testing can only show the presence of defects , never their absence. Ex: Dijkstra's Doctrine**
3. **Testing applies all through the software life cycle and is not end of cycle activity. Ex: Test in time:-**
4. **Understand the reason behind the test. Ex: The Cat and the Saint**
5. **Test the tests first. Ex: Patient & Doctor**
6. **Test Develop immunity and have to be revised constantly.**
   Ex: The Pesticide Paradox
7. **Defect occurs in convoys or clusters and testing should focus on these convoys Ex: The Convoy and the Rags**
8. **Testing encompasses defect prevention. Ex: The Policeman on the Bridge**
9. **Testing is a fine balance of defect prevention and a defect detection.**
   Ex: The Ends of the pendulum
10. **Intelligent and well planned automation is key to realizing the benefits of testing. Ex: Automation Syndrome**
11. **Testing requires talented, committed people who believe in themselves and work in teams Ex: Men in Black**

Arunai Engineering College

1. The Incomplete Car:-

Car Salesman:- " the car is complete – you just need to paint it "

Sales Representative/ Engineer:- "This Car has the best possible transmission and brake, and accelerate from 0 to 80mph in under 20 seconds!".
Customer: "Well that may be true, but unfortunately it accelerates (even faster) when I press the break pedal"

Above conversation concludes that the car is not tested properly , so customer finding the fault. "*Testing Should Focus on Finding Defects before Customers Find Them*"

2. Dijkstra's Doctrine:-

Consider a program (Dijkstra's Doctrine) that is supposed to accept a six character code and ensure that
the first character is numeric and rests of the characters are alphanumeric. How many combinations of input data should we test.

The first character can be filled up in one of 10 ways (the digits (0-9) . → 10
The Second through sixth characters can each be filled up in 62 ways(digits 0-9 (10) , lower case letters a-z (26) and capital letters A-Z (26)) → $(10+26+26)^5 \rightarrow 62^5$
This means that we have a total of 10 * (62^5) or 9,16,328,320 valid combinations of values to test.
Assuming that each combinations takes 10 seconds to test, testing all these valid combinations will take approximately 2905 years.

"Testing can only prove the presence of defects, never their absence"

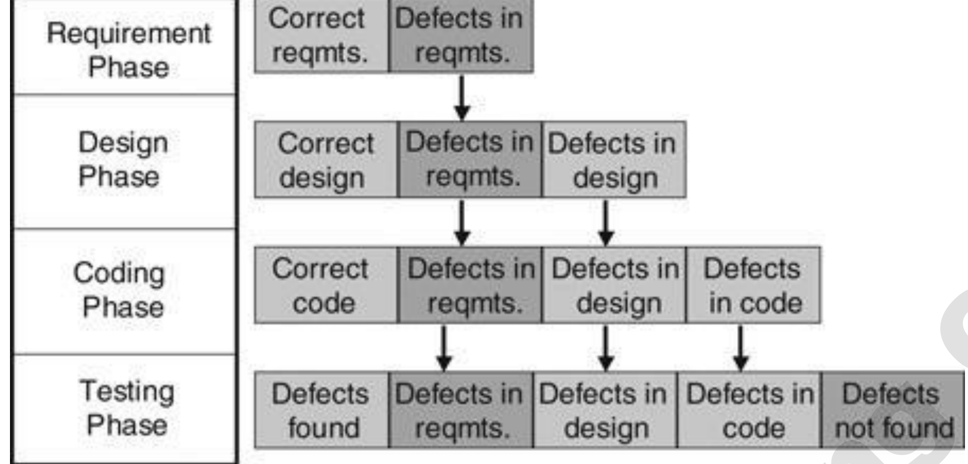
3. A Test in time:-

Defect in a product can come from any phase. There could have been errors while gathering initial requirements .If a wrong or incomplete requirement forms the basis for the design and development of a product, then that functionality can never be realized correctly in the eventual product. Similarly , when a product design – which forms the basis for the product development - is faulty, then the code that realizes the faulty design will also not meet the requirements. An Essential Conditions should be that

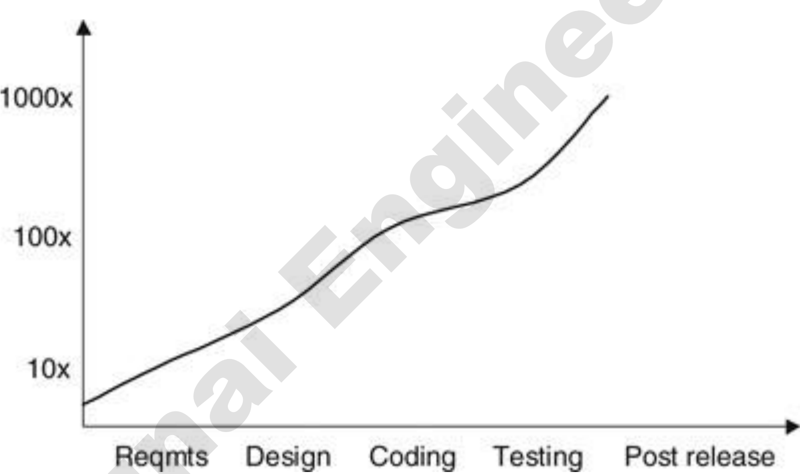
Arunai Engineering College

every phase of the software development (requirements, design, coding)should catch and correct defects at that phase, without letting the defects seep to the next stage. Organization incurs extra expenses for

- ☞ Performing a wrong design based on the wrong requirements;
- ☞ Transforming the wrong design into wrong code during the coding phase
- ☞ Testing to make sure the product complies with the (wrong requirements
- ☞ Releasing the product with the wrong functionality



How defects from early phases add to the costs.



Compounding effect of defects on software costs.

Arunai Engineering College

The cost of building a product and the number of defects in it increases steeply with the number of defects allowed to seep into the layer phases.

4. The Cat and the Saint

> A saint sat meditating. A cat that was prowling around was disturbing his concentration. Hence he asked his disciples to tie the cat to a pillar while he meditated . This sequence of events became a daily routine. The tradition continued over the years with the saint's descendents and the cat's descendents. One day, there were no cats on the hermitage. The disciples got panicky and searched for a cat, saying , " We need a car. Only when we get a cat, can we tie it to a pillar and only after that saint can start meditation !"

"Why one tests" it as important as " What to test " and "How to test ".

From the story , If we carry out tests without understanding why we are running them, we will end up in running inappropriate test that do not address what product should do.!

5. Test the Tests First :-

An audiologist was testing a patient , telling ,"I want to test the range within which you can hear . I will ask you from various distances to tell me your name, and you should tell me your name. Please turn back and answer." The patient understood what needs to be done

Doctor(from 30 feet): What is your name ?

……………..

Doctor (from 20 feet): What is your name ?

…………………………

Doctor (From 10 feet) What is your name ?

Patient : For the third time , let me repeat my name is Pushpa!.

From the above example it is important to make sure that the test themselves are not faulty before we start using them. One way of making sure that tests are tested is to document the inputs and expected outputs for a given test and have this description validated by an expert or get it counter checked by some means outside the tests themselves. " *Test the test first--- a defective test is more dangerous than defective products!"*

Annai Engineering College

6. The Pesticide Paradox:-

Pest gets used to new pesticides, develop immunity, and render the new pesticides ineffective. In subsequent years, the old pesticides have to be used to kill the pests which have not yet developed this immunity and new improved formulae that can combat these tougher variants of pests have to be introduced. This combination of new and old pesticides could sometimes even hinder the effectiveness of the (working)

Defects are like pests, testing is like designing the right pesticides to catch and kill the pests, and the test cases that are written are like pesticides .

Tests are like pesticides- you have to constantly revise their composition to tackle new pests(Defects) .

There are two possible ways to explain how products develop this "immunity" against test cases. The initial tests go a certain distance into the code and are stopped from proceeding, further because of the defects they encounter. Once these defects are fixed, the tests proceed further, encounter newer parts of the code that have been dealt with before, and uncover new defects.

A next way is that when the tester starts exercising a product, initial defects prevents them from using the full external functionality. As tests are run, defects are uncovered, and problems are fixed, users get to explore new functionality that has not been used before and this cause newer defect to exposed.

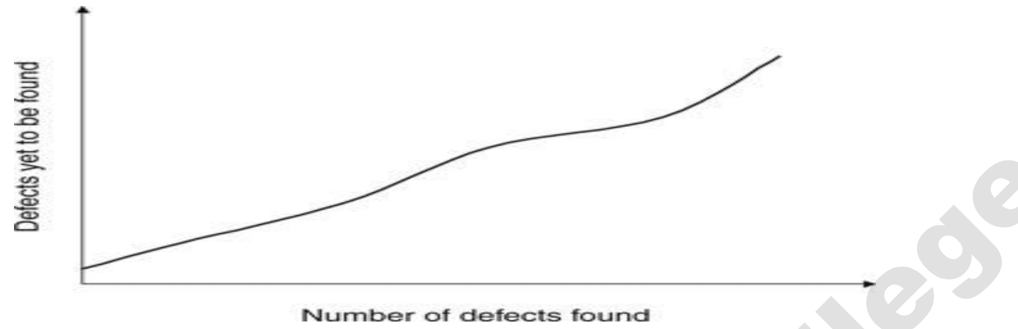
7. The Convoy and the Rags

All of us experience traffic congestions. Typically, during these congestions, we will see a convoy effect. There will be stretches of roads with very heavy congestions, with vehicles looking like they are going in a convoy. This will be followed by a stretch of smooth sailing (rather, driving) until we encounter the next convoy.

Testing can only find a part of defects that exist in a cluster; fixing a defect may introduce another defect to the cluster.

Annai Engineering College

A fix for a defect is made around certain lines of code. This fix can produce side-effects around the same piece of code. A fix for one defect generally introduces some instability and necessitates another fix. All these cause the convoy of defects in certain parts of the product. whenever a product undergoes any change, these error-prone areas need to be tested as they may get affected.



The number of defects yet to be found increases with the number of defects uncovered.

8. The Policeman on the Bridge:-

There was a bridge in a city. Whenever people walked over it, they would fall down. To take care of this problem, the city appointed a strong policeman to stand under the bridge to save people who fall down. While this helped the problem to some extent, people continued to fall down the bridge when the policeman was not around, or when he could not catch them properly.

When the policeman retired, a new policeman was appointed to the job. During the first few days, instead of standing at the bottom of the bridge and catching the falling people, the new policeman worked with an engineer and fixed the hole on the bridge, which had not been noticed by the earlier policeman. People then stopped falling down the bridge and the new policeman did not have anyone to catch.

Like the second police officer in the above story, they know people fall and they know why people fall. Rather than simply catch people who fall (and thereby be exposed to the risk of a missed catch), they should also look at the root cause for falling and advise preventive action. It may not be possible for testers themselves to carry out preventive action. ***"Prevention is better than cure—you may be able to expand your horizon much farther."*** Defect prevention is a part of a tester's job.

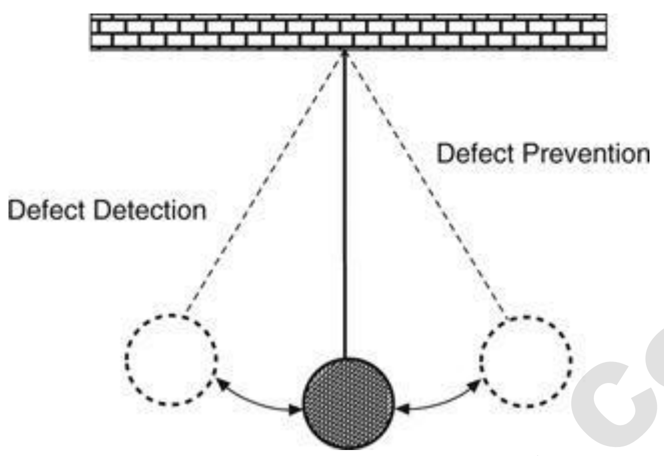
9. The Ends of the Pendulum :-

The eventual goal of any software organization is to ensure that the customers get products that are reasonably free of defects. There are two approaches to achieving this goal.

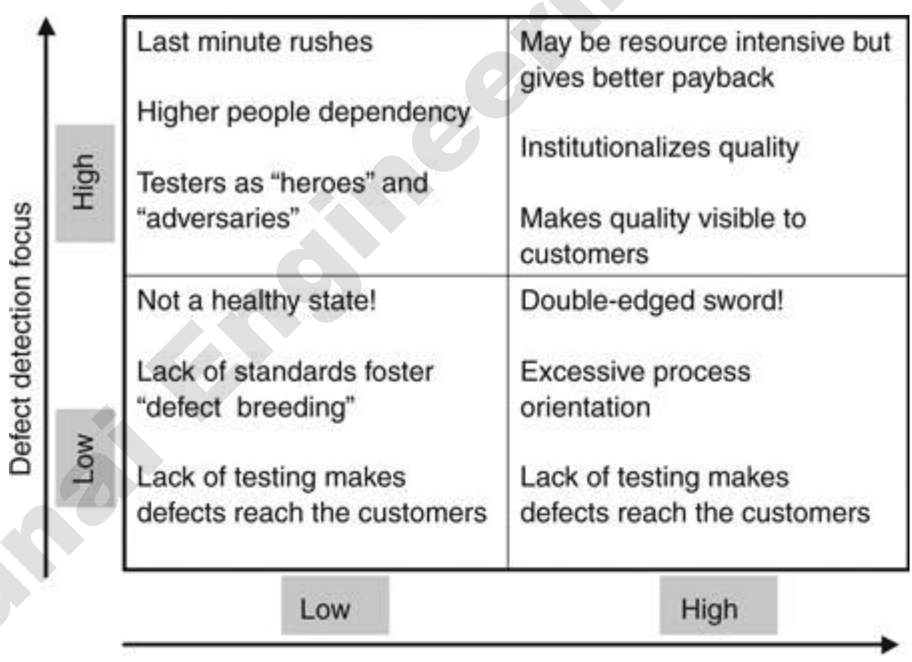
1) focus on defect detection and correction
2) focus on defect prevention.

Arunai Engineering College

These are also called quality control focus and quality assurance focus. Testing is traditionally considered as a quality control activity, with an emphasis on defect detection and correction. Quality assurance is normally associated with process models such as CMM, CMMI, ISO 9001, and so on. Organizations view these two functions as mutually exclusive, "either-or" choices.



Quality control and quality assurance as two methods to achieve quality.



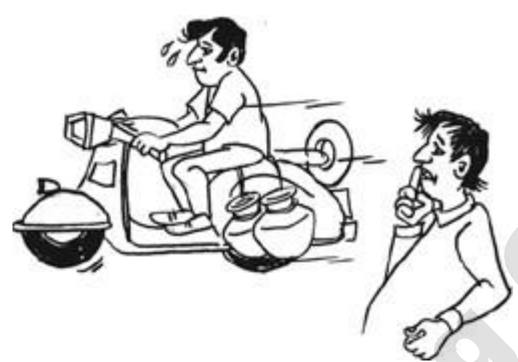
Arunai Engineering College

10.Automation Syndrome:-

A farmer had to use water from a well which was located more than a mile away.

Crop Cycle	Farmer's Approach	Outcome
1st	• 100 people to draw water from the well • pot of water a day	crops failed

2nd	• thought about automation to increase productivity and to avoid failures • bought 50 motorcycles, laid off 50 of his workers • asked each rider to get two pots of water. • process of learning to balance the motorcycles, the number of pots of water they could fetch fell.	crops failed again.
3rd	• all workers were laid off except one • bought a truck to fetch water • realized the need for training and got his worker to learn driving. • Road was narrow, truck did not help in bringing in the water	crops failed again.
After these experiences the farmer said, "My life was better without automation!"		



The moral of the above story as it applies to testing is that automation requires careful planning, evaluation, and training. Automation may not produce immediate returns.

Some of the points that should be kept in mind while harping on automation are as follows.

- Know first why you want to automate and what you want to automate, before recommending automation for automation's sake.
- Evaluate multiple tools before choosing one as being most appropriate for your need.
- Try to choose tools to match your needs, rather than changing your needs to match the tool's capabilities.
- Train people first before expecting them to be productive.
- Do not expect overnight returns from automation.

<u>11.Men in Black:-</u>

The testing team was seeded with motivated people who were "free from cognitive conflict that hampers developers when testing their own programs." The team was given an identity (by a black dress, amidst the traditionally dressed remainder of the organization) and tremendous importance. All this increased their pride in work and made their performance grow by leaps and bounds, "almost like magic." Long after the individual founding members left and were replaced by new people, the "Black Team" continued its existence and reputation. The biggest bottleneck in taking up testing as a

Arunai Engineering College

profession is the lack of self-belief. This lack of self-belief and apparent distrust of the existence of career options in testing makes people view the profession as a launching pad to do other software functions. As a result, testers do not necessarily seek a career path in testing and develop skepticism towards the profession.

The Tester's Role in a Software Development Organization:-

- Testing is sometimes erroneously viewed as a destructive activity.
- The testers job is to reveal defects, fins weak points, inconsistent behavior, and circumstances where the software does not work as expected.
- Given the nature of the testers task , it is difficult for developers to effectively test their own code
- **Effective Tester:** To be most effective as a tester requires extensive programming experience in order to understand how code is constructed, and where and what kind of, defects are likely to occur.
- **Goal as a tester** is to work with the developers to produce high quality software that meets customer's requirements
- Projects should have an appropriate developer /tester ratio. The ratio will vary depending on available resources, type of projects and TMM level (Testing Maturity Model)

  Example: - An embedded real time system needs to have a lower developer /tester ratio. **(2/1)** than a simple data base application (4/1) .At higher TMM levels where there is a well defined testing group, the developer/tester ratio would tend to be on the lower end(2/1 versus 4/1) because of the availability of tester resources.
- Cooperation with the code developers, tester also need to work along side with requirement engineers to ensure that requirement are testable, ant to plan for system and acceptance test(client are also involved in the latter part).
- Testers also need to work with designers to plan for integration and unit test.
- Test managers will need to cooperate with project managers in order to develop reasonable test plans, and with upper management to provide input for the development and maintenance of organizational testing standards , policies and goals.
- Testers also need to corporate software quality assurance staff and software engineering process group members.
- In view of these requirements for multiple working relationships, communication and extreme working skills are necessary for a successful career as a tester.

If you are employed by an organization that is assessed at **TMM levels 1 or 2**

Arunai Engineering College

you may find that there is no independent software test function in the organization, so the testers in this case may be a part of the development group, but with special assignment to testing, or they may be apart of the software quality assurance group.

TMM level 3 and higher of the TMM

the testers may not necessarily belong to a independent organizational entity, testers should always have managerial independence from developers in the TMM level 3.

Testers are specialist their main function is

- to plan
- execute,
- record
- analyze tests. They do not debug software.

When defects are detected using testing, software should be returned to the developers who locate the defect and repair the code

Testers need the support of management. Developers , analysts and marketing staff need to realize that tester add value to a software products in that they detect defects and evaluate quality as early as possible in the software life cycle. Tester need to have a positive view of their work. Management must support them in their effort and recognize their contribution to the organization.

Origins of Defects:-

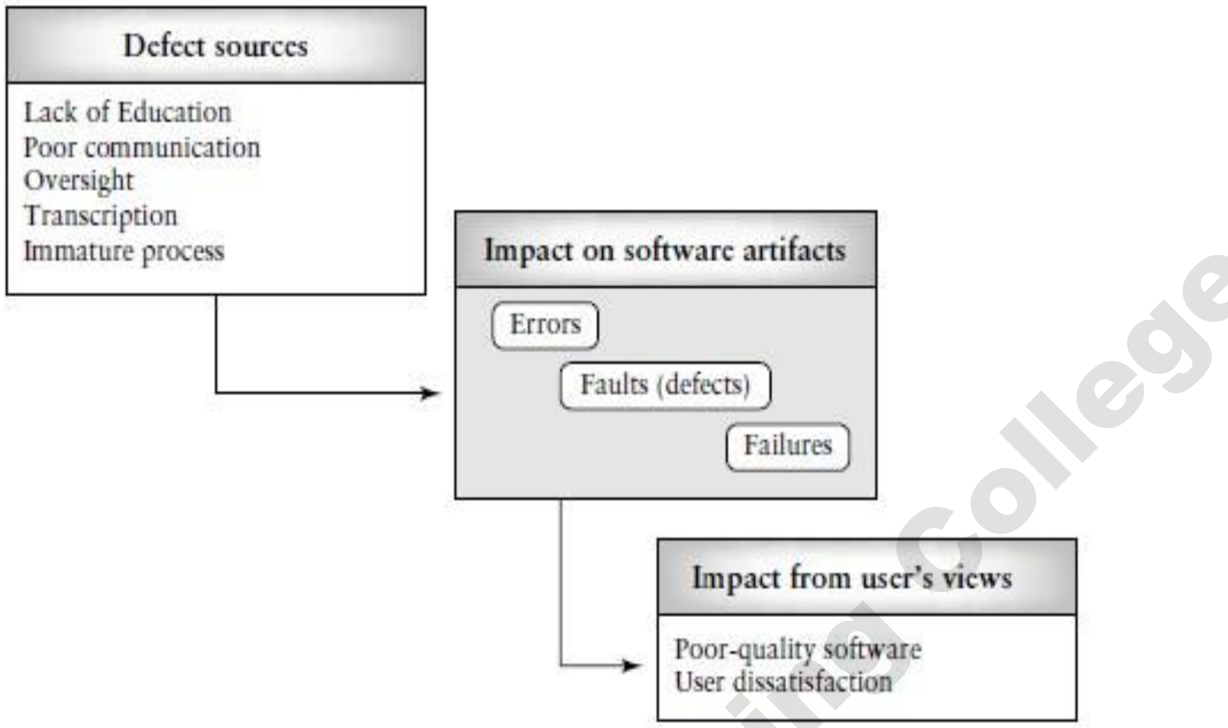
Defects have harmful effects on software users, and software engineers work very hard to produce high quality software with a low numbers of defects.

Reason for Defects are shown below

1. **Education :-** The software engineer did not have the proper educational background to prepare the software artifacts. They did not understand how to do something. For example, a software engineer who did not understand the precedence order of operators in a particular programming language could inject a defect in an equation that uses the operators for calculation.
2. **Communication:-** The software engineer was not informed abut something by a colleague. For example , if engineer 1 and engineer2 working on interfacing modules , and engineer 1 doesn't inform engineer2 that no error checking code will appear in the interfacing module he is developing , engineer 2 might have an incorrect assumption relating to the presence /absence of an error check and defects will result.
3. **Oversight :-** The software engineer omitted to do something .For example a software engineer might omit an initialization statement
4. **Transcription:-** The software engineer knows what to do, but makes a mistake in doing it. A simple example is a variable name being misspelled when entering the code.

Arunai Engineering College

5. **Process:**- The process used by the software engineer misdirected the action. For example, a development process that did not allow sufficient time for a detailed specification to be developed and reviewed could lead to specification defects.



- Goal as tester is to discover defects preferably before the software is in operation. One of the way we do this is by designing test cases that have a high probability of revealing defects.
- In the experimental scenario
  - a tester develops hypotheses about possible defects (Principle 2 and Principle 9)
  - Test cases are then designed based on the hypotheses.
  - The tests are run and results analyzed to prove, or disprove the hypotheses.
- Myers has a similar approach to testing. He describes the successful test as one that reveals the presence of Hypotheses defect.
- He compares the role of a tester as a doctor who is in the process of constructing a diagnosis for an ill patient. The doctor develops hypotheses about possible illness using her knowledge of possible diseases, and the patients symptoms. Test are made in order to make the correct diagnosis.
- A successful test will reveal the problem and the doctor can being the treatment. Completing the analogy of doctor and ill patient, one could view defective software as the ill patient. Testers as doctors need to have knowledge about possible defects (illness) in order tom develop defect hypotheses. They use the hypotheses to:-
  - Design Test cases;
  - Design Test procedures;

Arunai Engineering College

- Assemble test sets;
- Select the testing levels(unit, integration,etc) appropriate for the tests;
- Evaluate the result of the tests;

Very useful concept related to defects, testing, and diagnosis is that of the fault model.

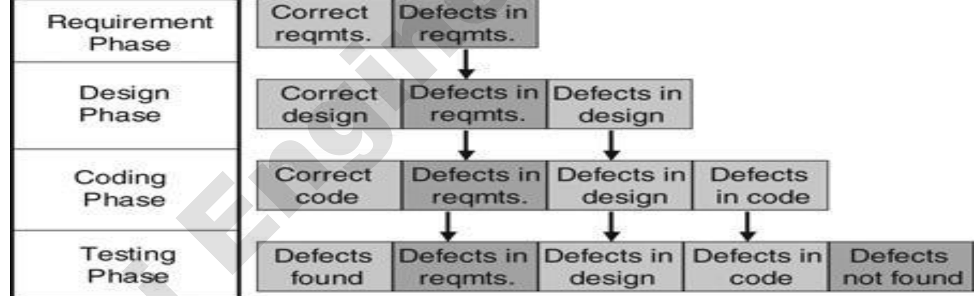
- A fault (defect) model can be described as a link between the error made(eg., a missing requirement, a misunderstood design elements, a typographical error) and the fault/ defect in the software.

Example of fault model a software engineer might have in memory is "an incorrect value for a variable was observed because the precedence order for the arithmetic operators used to calculate its value was incorrect"→ this could be called as "incorrect operator precedence operator".

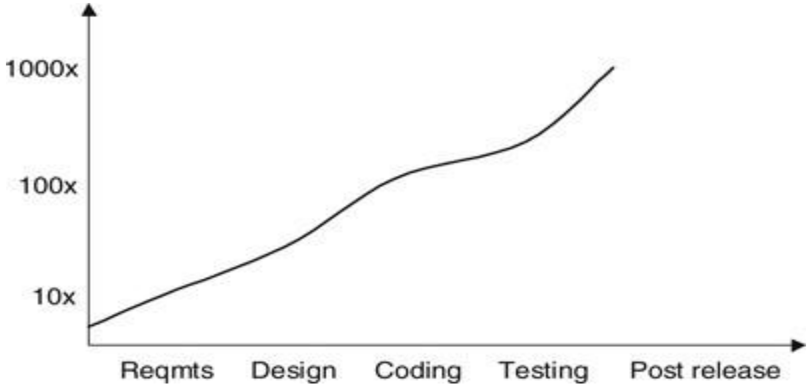
Cost of Defect

Organization incurs extra expenses for

- ☞ Performing a wrong design based on the wrong requirements;
- ☞ Transforming the wrong design into wrong code during the coding phase
- ☞ Testing to make sure the product complies with the (wrong requirements
- ☞ Releasing the product with the wrong functionality



How defects from early phases add to the costs.



Compounding effect of defects on software costs.

Arunai Engineering College

The cost of building a product and the number of defects in it increases steeply with the number of defects allowed to seep into the layer phases.

DEFECT CLASSES:-

Defect can be classified in many ways. It is important for an organization to adapt a single classification scheme and apply it to all projects. Developers , testers and SQA staff should try to be as consistent as possible when recording defect data

Defects are assigned to four major classes reflecting their point of origin in the software life cycle- the development phases in which they were injected. These classes are:-

- **Requirements\ Specifications**
- **Design**
- **Code**
- **Testing**

Requirements and Specification Defects:-

The beginning of software life cycle is critical for ensuring high quality in the software being developed. Defects injects in early phases can persist and be very difficult to remove in later phases. Since many requirements are written unnatural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant, and imprecise requirements.

1. **Functional Description Defects:-**

The overall description of what the product does, and how it should behave (Input/Output), is incorrect, ambiguous, and /or incomplete.

2. **Feature Defects**

Features may be described as distinguishing characteristics of a software component or system .Features refers to functional aspects of software that map to functional requirement described by the user and the client, it also maps quality such as performance and reliability. Feature defects are mainly due to features description that are missing, incorrect, incomplete or superfluous.,

3. **Feature Interaction Defects:-**

Mainly due to incorrect description of how the features should interact. For ex:- suppose one features of a software system supports adding a new customer to a customer database. This feature interacts with another feature that categorizes the new customer. Classification feature impact on where the storage algorithm places the new customer in the database, and also affects another feature that periodically support sending advertising information to customers in a specific category.

4. **Interface Description Defects:-**

Description of how the target software is to interface with external software, hardware and users. For detecting many functional description defects, black box

Arunai Engineering College

testing techniques, which are based on functional specification of the software, offer the best approach. Black Box testing techniques such as equivalence class partitioning, boundary value analysis, state transition testing, and cause and effect graphing, which are useful defecting functional type of defects.

Black Box based tests can be planned at the unit, integration, system and acceptance levels to detect requirements/specification defects.

DESIGN DEFECTS:-

Design defects occur when system components, interactions between system components, interaction between the components and outside software /hardware, or users are incorrectly designed. Design of algorithm, control, logic, data elements, module interface description, and external software/ hardware/ user interface description. When describing these defects we assume that the detailed design description for the software modules is the pseudo code level with processing steps

1. **Algorithmic and Processing Defects:-**

These occur when the processing steps in the algorithm as described by the pseudo code are incorrect. Eg:- the pseudo code may contain a calculation that is incorrectly specified, or the processing steps in the algorithm written in pseudo code language may not be in the correct order.

Letter case a step may be missing or steps may be duplicated. Example of a defect in this sub class is the omission of error condition checks such as division by zero.

2. **Control, Logic and Sequence Defects:-**

Control defect occur when logic flow in the pseudo code is not correct. For example , branching to soon, branching to late, or use of an incorrect branching, condition. Other examples in this subclasses are unreachable pseudo code elements, improper nesting, improper procedure or function calls. Logic defects usually relate to incorrect use of logic operators, such as <,>

3. **Data Defects:-**

These are associated with incorrect design of data structures. For example a record may be lacking a field, an incorrect type is assigned to a variable or field in a record, an array may not have the proper number of elements assigned , or storage space may be allocated incorrectly.

4. **Module Interface Description Defects :-**

These are defects derived from , for example , using incorrect, and/or inconsistent parameter type, an incorrect number of parameters, or an incorrect ordering of parameters

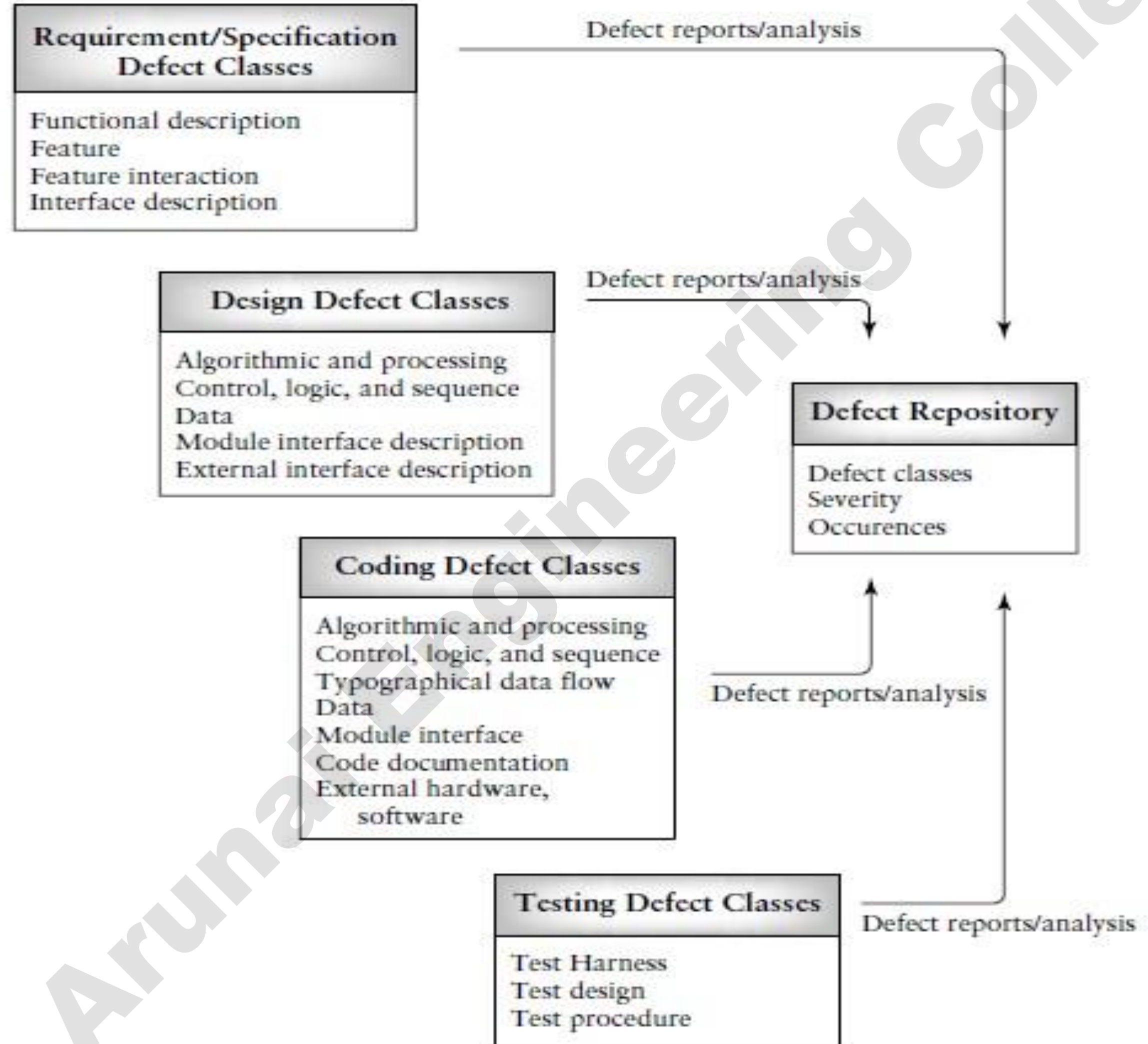
Arunai Engineering College

5. **Functional Description Defects:-**

The defect in this category include incorrect, missing, and/or unclear design element . Eg the design may not properly describe the correct functionality of a module.

6. **External Interface Description Defects:-**

These are derive from in correct design description for interfaces with COTS components, external software systems, databases and hardware devices(eg:-I/O devices).Other example are user interface description defects where there are missing or improper commands, improper sequence of commands, lack of proper message, and/or lack of feedback message for the users.



Arunai Engineering College

CODING DEFECTS:-

Coding Defects are derived from error in implementing the code, coding defects are closely related to design classes especially if pseudo code has been used for detailed design. Some coding defects come from a failure to understand programming language constructs, and miscommunication with the designer.

1. **Algorithm and Processing Defects:-**

Adding levels of programming detail to design , code related algorithmic and processing defect would now include unchecked overflow and underflow conditions, comparing inappropriate data types, converting one data type to an other , in correct ordering of arithmetic operators, misuse or omission of parenthesis, precision loss and incorrect use of signs.

2. **Control, Logic and Sequence Defects :-**

On the coding level these would include incorrect expression of case statements, incorrect iteration of loops and missing paths

3. **Typographical Defects:-**

These are principally syntax errors, for example incorrect spelling of variable name, that are usually detected by compiler, self reviews, or peer reviews

4. **Initialization Defects:-**

These occur when initialization statements are omitted or are incorrect. This may occur because of misunderstanding or lack of communication between programmers, and /or programmers and designers, carelessness, or misunderstanding of programming environment.

5. **Data Flow Defects:-**

There are certain reasonable operational sequences that data should flow through. For example a variable should be initialized, before it is used in a calculation or a condition. It should not be initialized twice before there is an intermediate use. A variable should not be disregarded before it is used.

6. **Data Defects :-**

These are indicated by incorrect implementation of data structures. For example , the programmer may omit a filed in a record, an incorrect type or access is assigned to a file, an array may not be allocated the proper number of elements.

7. **Module Interface Defects:-**

As in the case of module design elements, interface defects in the code may be due to using in correct or inconsistent parameter types, an incorrect number of parameters, or improper ordering of the parameters, improper design, programmer may implement an incorrect sequence of calls or calls to nonexistent modules

8. **External Hardware, Software Interface Defects:-**

These defects arise form problems related to system calls, links to databases, input/output sequences, memory usage, resource usage, interrupts and exception handling , data exchange with hardware, protocols formats, interface with build files, and timing sequences

Arunai Engineering College

TESTING DEFECTS:-

Defects are not confined to code and its related artifacts. Test plans, test cases, test harnesses, and test procedures can also contain defects. Defects in test plans are best detected using review techniques.

1. **Test Harness Defect:-**

In order to test software, especially at the unit and integration levels, auxiliary code must be developed. This is called the test harness or **scaffolding** code. Test harness code should be carefully designed, implements and testes since it a work product and much of this code can be reused when the new release of the software are developed.

2. **Test Case Design and Test Procedure Defects:-**

These would encompass incorrect, incomplete, missing , inappropriate test cases, and test procedures. These defects are again best detected in test plan reviews. Defects are revealed during the testing process itself by means of a careful analysis of test conditions and test results. Repairs will then have to be made.

DEFECT EXAMPLES:- The Coin Problem Requirement Specification

Specification for program calculate_coin_value

This program calculates the total dollars and cents value for a set of coins. the user inputs the amount of pennies, nickels , dimes , quarters, half-dollars, and dollar coins held. There are six different denominations of coins. The program outputs the total dollar and cent values of the coins to the users

Input:number_of_coins is an integer
Outputs:- number_of_dollars is an integer
Number_of_cents is an integer

A spec above shows the sample informal specification for a simple program that calculates the total money value of a set of coins. The program could be a component of an incentive cash register system to support retail store clerks.

Coin Problem in Detail : (100 cent = 1 dollar)

if suppose input for coin values given as 1 for all then the calculation as shown below.

No of Coins (input)		Coin Value		
1(pennies)	X	1	=	1
1 (nickels)	X	5	=	5
1(dimes)	X	10	=	10
1(quarters)	X	25	=	25
1(half-dollars)	X	50	=	50
1(dollar)	X	100	=	100
				191

Arunai Engineering College

Output

No of Dollars : 1

No of Cents : 91

The given specification does not specify the above details clearly.

Requirements/ specifications defects,

1) Functional description defects

a) No of coins, dollars , cents > 0

pre & post conditions are helpful

b) In each denomination – largest number allowed is missing

upper limit for cents & dollars - not given

2) Interface description defect

how user interact to provide i/p , o/p

Explanation

1) Functional Description defects arise because the functional description is ambiguous and incomplete .It does not state that the input, number_of_coins, and the output, number_of_dollars and number_of_cents, should all have values of zero or greater. The number_of_coins cannot be negative and the values in dollars and cents cannot be negative in the real world domain .

Formally stated set of [preconditions and post conditions would be helpful here, and would address some of the problem with the specification. These are also useful for designing black box tests.

A precondition is a condition that must be true in order for a software component to operate properly.

In this case a useful precondition would be one that states for example ,
Number_of_coins >=0

A Post condition is a condition that must be true when a software component completes its operation properly.

A useful post condition would be :-

Number_of _dollars, number_of_cents >=0

2) Interface Description defects

It is not clear from the specification how the user interacts with the program to provide input and how the output is to be reported. Because of ambiguities in the user interaction description the software may be difficult to use.

Arunai Engineering College

Design of Coin Problem

```
Design Description for Program Calculate_coin_values

Program calculate_coin_values
number_of_coins is integer
total _coin_value is integer
number_of_dollars is integer
number_of_cents is integer
coin_value is array of six integers representing each coins value in cents
initialized to 1,5,10,25,25,100
begin

initialize total_coin_value to zero
initialize loop_counter to one
wjile loop_counter is less than six
begin
     output "enter number of coins"
     read(number_of_coins)
total_coin_value=total_coin_value+ number_of_coins * coin_value[loop_counter]
increment loop_counter

end
number_dollars=total_coin_value/100
number_of_cents=total_coin_value-100 * number_of_dollars
output (number_of_dollars, number_of_cents)
end
```

Design Defect

1) Control Logic & Sequencing defect → while counter
2) Algorithmic & Processing Defect → invalid i/p value not checked
3) Data Defect → array value 25 - 2 times
4) External Interface Description defect → Order of i/p , when to stop , help msg , feedback not given

Explanation

1) **Control, Logic and Sequencing Defects:-**

The defect in this subclass arises from an incorrect "while" loop condition(should be less than or equal to six)

2) **Algorithmic and Processing Defects:-**

These arise from the lack of error checks for incorrect and /or invalid inputs, lack of path where users can correct erroneous inputs, lack of a path for recovery from input errors. The lack of an error check could be counted as functional design defects since the design does not adequately describe the proper functionality for the program.

Arunai Engineering College

3) Data Defects:-

This defect relates to an incorrect value for one of the elements of the integer array, coin_values, which should read 1,5,10,25,50,100

4) External Interface Description Defects:-

These are defects arising from the absence of input messages or prompts that introduced the program to the user and request inputs. The user has no way of knowing in which order the number of coins for each denomination must be input, and when to stop inputting values. There is absence of help message and feedback for user if he wishes to change an input or learn the correct format and order for inputting the number of coins.

The control and logic design defects are best addressed by white box based tests,(condition/Branch testing, loop testing).

The program below is a C like programming language. Without effectives reviews the specification and design defects could propagate to the code. Here additional defects have introduced in the coding phase.

Coin Problem Coding :

```
/*********** Program calculate_coin_values caluates the dollar and cents value
of a set of coins of different dominations input by the user denominations are
pennies, nickels, dimes ,quarters, half dollars and dollars **************/

main()
int  total_coin_value;
int number_of_coin=0;
int number_of_dollar=0;
int number_of_cents=0;
int coin_value={1,5,10,15,25,25,30};
{
      int  i=1;
      while (i  < 6 )
             {
                    printf("input number of coins\n");
                    scanf("%d", number_of_coins);
      total_coin_value=total_coin_value+(number_of_coins *     coin_value{i]);
}
      i=i+1;
      number_of_dollars=total_coin_value/100;
      number_of_cents=total_coin_value-(100*number _of_dollars);
      printf("%d\n", number _of_dollars);
      printf("%d\n",number_of_cents);
}
/*******************************/
```

Arunai Engineering College

Coin Problem Coding Defect :

1) Data flow defect → total_coin _value not initialized
2) Data Defect → Array value 25 - 2 times
3) Control , logic and sequence Defect → While(i<6)
4) External Interface Description defect → scanf without &
5) Algorithmic and Processing Defects
6) Code Documentation Defect

1) **Data Flow Defects:-**

The variable total_coin_value is not initialized. It is used before it is defined.

2) **Data Defects:-**

The error in initializing the array coin_values is carried over form design and should be counted as design defect.

3) **Control , Logic and Sequence Defects:-**

These include the loop variable increment step which is out of the scope of the loop. Note that incorrect loop condition(i<6) is carried over from design and should be counted as a design defects

4) **External Hardware , Software Interface Defects:-**

The call to the external function "scanf" is incorrect. The address of the variable must be provided (&number_of_coins)

5) **Algorithmic and Processing Defects:-**

The division operator may cause problems if negative values are divided, although this problem could be eliminated with an input check.

6) **Code Documentation Defects:-**

The documentation that accompanies this code is incomplete and ambiguous. It reflects the deficiencies in the external interface description and other defects that occurred during the specification and design. Vital information is missing for anyone who will need to repair, maintain or reuse this code.

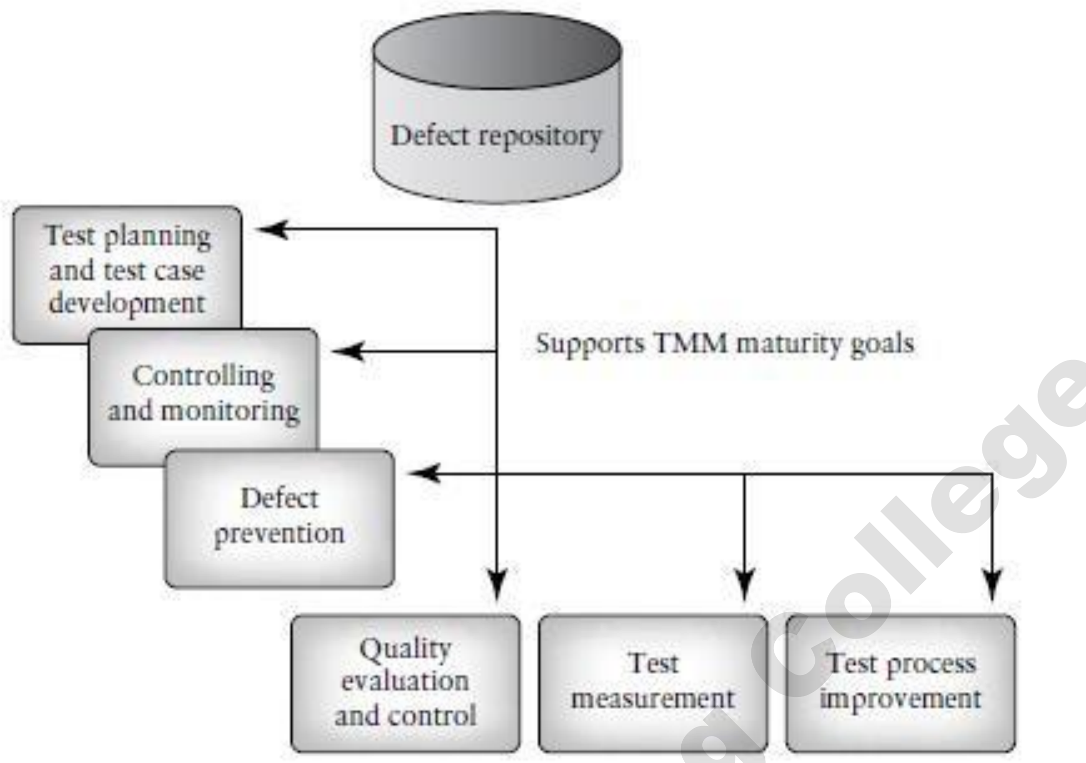
The control, logic and sequence, data flow defects found in this sample could be detected by combination of white and black box testing techniques.

Black Box tests may work well to reveal the algorithmic and data defects. the code documentation defects require a code review for detection. The external software interface defects would probably be caught by a good complier.

Poor quality of this small program is due to defects injected during several of the life cycle phases with probable causes ranging from lack of education, a poor process, to oversight on the part of the designers and developers.

Arunai Engineering College

DEVELOPER/ TESTER SUPPORT FOR DEVELOPING A DEFECT REPOSITORY :-



It is important if you are a member of a test organization to illustrate to management and colleagues the benefit of developing a defect repository to store defect information. Software Engineers and Test Specialists we should follow the example of engineers in other disciplines who realized the usefulness of defect data. Defect monitoring should continue for each ongoing projects. The distribution of defects will change as you make changes in your processes. The defect data is useful for test planning, a TMM level 2 maturity goals. It helps you to select applicable testing techniques, design and the test cases you need and allocate the amount of resources you will need to devote to detecting and removing these defects.

A defect repository can help to support achievements and continuous implementation of several TMM maturity goals including controlling and monitoring of test, software quality evaluation and control ,test measurements, and test process improvement.

Arunai Engineering College

ARUNAI ENGINEERING COLLEGE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
IT8076-SOFTWARE TESTING (2017 Regulation)

UNIT II TEST CASE DESIGN

Test case Design Strategies – Using Black Box Approach to Test Case Design – Boundary Value Analysis – Equivalence Class Partitioning – State based testing – Cause-effect graphing – Compatibility testing – user documentation testing – domain testing - Random Testing – Requirements based testing – Using White Box Approach to Test design – Test Adequacy Criteria – static testing vs. structural testing – code functional testing – Coverage and Control Flow Graphs –Covering Code Logic – Paths – code complexity testing – Additional White box testing approaches Evaluating Test Adequacy Criteria.

TEST CASE DESIGN STRATEGIES

- Develop effective test cases for execution based testing.
- positive consequences of effective test cases
  - A greater probability of detecting defects
  - A more efficient use of organizational resources
  - A higher probability for test reuse
  - Closer adherence to testing and project schedules and budgets
  - The possibility for delivery of higher quality software products

The two basic testing strategies

Test Strategy & Tester's View	Knowledge Sources	Methods
Black Box Inputs Outputs	Requirements Documents Specification Domain Knowledge Defect Analysis Data	Equivalence class Partitioning Boundary value analysis State transition testing Cause and Effect Graphing Error guessing
White Box	High level Design Detailed Design Control Flow Graphs Cyclomatic Complexity	Statement Testing Branch Testing Path Testing Data Flow testing Mutation Testing Loop Testing

Black Box Testing

- size of the software -> simple module, member function, or object cluster to a subsystem or a complete software system.The description behavior or functionality for the software under test may come from a formal specification

Arunai Engineering College

an input/ process output diagram (IP0), or well defined set of pre and post conditions.

- Because the black box approach only considers software behavior and functionality, it is often called functional or specification based testing.
- This approach is useful for revealing requirements and specification defects.

White Box Approach

- Since designing, executing and analyzing the results of white box testing is very time consuming, this strategy is usually applied to smaller sized pieces of software such as module or member function .
- White box testing methods are especially useful for revealing design and code based control, logic and sequence defects, initialization defects and data flow defects.

Using the Black Box Approach to Test Case Design:-

1)Equivalence Class Partitioning:-

Partition the input domain of the software into valid and invalid classes. Invalid classes represent erroneous or unexpected inputs.

Advantage:-

- exhaustive testing - eliminated
- selecting a subset of test inputs with a high probability of detecting a defect
- cover a larger domain of inputs/outputs with a smaller subset selected from an equivalence class.

Guidelines

Input Conditions	**no of equivalent classes**	**EXAMPLE**
range of values	one valid & two invalid classes	Eg: range of 1-499 Valid -> all values from 1-99 Invalid -> values < 1 Invalid → values > 499
specific value	one valid & two invalid classes	Eg:- If the specification for a Product code(3115) , Valid -> valid Product code {3115} Invalid -> valid Product code<31→15 Invalid valid Product
Members of a set	one valid & one invalid classes	eg:- paint module states that the Color RED, BLUE, GREEN and YELLOW are allowed as inputs Valid -> RED Invalid -> BLACK
must be condition (Boolean)	one valid & one invalid classes	Eg:- if a specification for a module states that the first character of a part identifier must be a letter Valid -> TOTAL Invalid -> 3PI
If the input specification in an equivalence class is not handled in an identical way by the software under test, then the class should be further partitioned into smaller equivalence classes		

Arunai Engineering College

Example: A specification of a square root function.

Function square_root
message (x:real)
when x >0.0
reply (y:real)
where y >0.0 & approximately (y*y,x)
otherwise reply exception imaginary_square_root
end function

input: x (4)
output : y (2) → Square root of x

I) Test Condition Relevant to Input Conditions:

1)The input conditions → variable x must be a real number and be equal to or greater than 0.0.

2) The output conditions →y must be a real number equal to or greater than 0.0, whose square is approximately equal to x.

II) Generate equivalence classes

EC1. The input variable x is real, valid.
EC2. The input variable x is not real, invalid.
EC3. The value of x is greater than 0.0, valid.
EC4. The value of x is less than 0.0, invalid.

III) equivalence class reporting Table (EC table)

Condition	Valid EC	Invalid EC
1	EC1 , EC3	EC2 , EC4

IV) Summary of Test I/ps using EC Partitioning

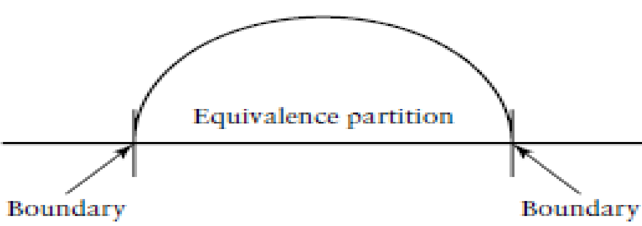
Test case Id	**i/p**	**Valid EC**	**Invalid EC**
TC1	-3	-	EC4
TC2	4.0	EC1,EC3	-
TC3	AB	-	EC2
TC4	-6.2	-	EC4

Provide testcases for all ECs present in EC table

2)Boundary Value Analysis :

- The test cases developed based on equivalence class partitioning can be strengthened by use of an another technique called boundary value analysis.
- boundary value analysis requires that the tester select elements close to the edges, so that both the upper and lower edges of an equivalence class are covered by test cases.

Arunai Engineering College



Guidelines

Input Conditions	Valid & Invalid Test Case	EXAMPLE
range of values	valid test cases→ ends of the range, invalid test cases → above and below the end of the range.	Eg: range between -1.0 and +1.0 input values of -1.0, -1.1, and 1.0, 1.1.
number of values	valid test cases → min & Max Numbers Invalid Test Case → Min-1 , max+1 numbers	Ex: house can have one to four owners 0,1 owners and 4,5 owners
ordered set,	focus on the first and last elements of the set.	i/p: {25,27,28} last element → 27,**28**,29 first element → 24,**25**,26

Example1:

The input specification for the module states that a widget identifier should consist of 3–15 alphanumeric characters of which the first two must be letters.

I) conditions that apply to the input:

(i) it must consist of alphanumeric characters,
(ii) the range for the total number of characters is between 3 and 15, and,
(iii) the first two characters must be letters.

II) Generate bounds groups

BLB—a value just below the lower bound
LB—the value on the lower boundary ALB—
a value just above the lower boundary
BUB—a value just below the upper bound
UB—the value on the upper bound
AUB—a value just above the upper bound

For our example module the values for the bounds groups are:
BLB—2 BUB—14
LB— 3 UB— 15
ALB—4 AUB—16

Generate Equivalent Classes

Condition1:

EC1. Part name is alphanumeric, valid.
EC2. Part name is not alphanumeric, invalid.

Arunai Engineering College

Condition2:
EC3. The widget identifier has between 3 and 15 characters, valid.
EC4. The widget identifier has less than 3 characters, invalid.
EC5. The widget identifier has greater than 15 characters, invalid.

Condition3:
EC6. The first 2 characters are letters, valid.
EC7. The first 2 characters are not letters, invalid.

III) Equivalence class reporting table.

Condition	Valid equivalence classes	Invalid equivalence classes
1	EC1	EC2
2	EC3	EC4, EC5
3	EC6	EC7

IV) Summary of Test Inputs using equivalence class & BVA

Test case identifier	Input values	Valid equivalence classes and bounds covered	Invalid equivalence classes and bounds covered
1	abc1	EC1, EC3(ALB) EC6	
2	ab1	EC1, EC3(LB), EC6	
3	abcdef123456789	EC1, EC3 (UB) EC6	
4	abcde123456789	EC1, EC3 (BUB) EC6	
5	abc*	EC3(ALB), EC6	EC2
6	ab	EC1, EC6	EC4(BLB)
7	abcdefg123456789	EC1, EC6	EC5(AUB)
8	a123	EC1, EC3 (ALB)	EC7
9	abcdef123	EC1, EC3, EC6 (typical case)	

Provide testcases for all ECs present in EC table and bound groups.

Example 2: Pin number input of ATM SYSTEM

Case Study : Apply ECP & BVA for pinno of ATM System

Example:

The Pinno input has following specification

I) Derive Input conditions

a) Only digits for pin no input
b) Values range from 0000 to 9999 (Length is 4)

Arunai Engineering College

II) Generate bounds groups

For our example module the values for the bounds groups are:

BLB— -1 BUB— 9998

LB— 0 UB— 9999

ALB—1 AUB— 10000

Generate Equivalent Classes

Condition1:

EC1.pinno i/p - only digits, valid.

EC2. pinno i/p with digits and other symbols , invalid.

Condition2:

EC3. The pinno i/p has value between 0000 and 9999 , valid.

EC4. The pinno i/p has value < 0000, invalid.

EC5. The pinno i/p has value > 9999, invalid.

III) Equivalane Class Report

i/p condition	valid EC	Invalid EC
1	EC1	EC2
2	EC3	EC4, EC5

IV) Summary of test i/ps

Test Case ID	Input Values	Valid EC & Bounds Covered	InValid EC & Bounds Covered
1.	9999	EC1 , EC3(UB)	-
2.	9998	EC1 , EC3(BUB)	-
3.	0000	EC1 , EC3(LB)	-
4.	0001	EC1 , EC3(ALB)	-
5.	W236	-	EC2
6.	-875	-	EC2 , EC4(BLB)
7.	10000	EC1	EC5(AUB)

3)State based Testing

Graph based testing methods are applicable to generate test cases for state machines such as language translators , work flows , transaction flows and data flows.

It is useful in

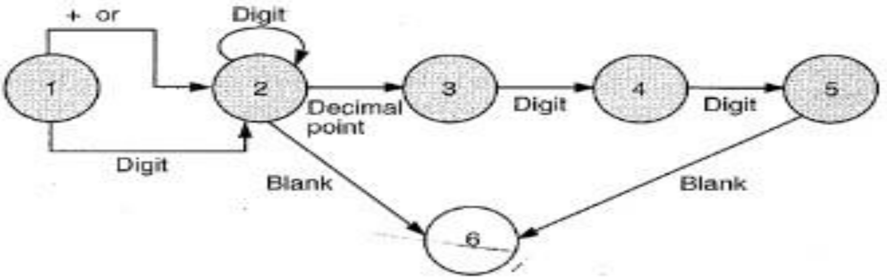
- product is language processor
- work flow modeling
- dataflow modeling

Example: validate number using simple rules (for language processor)

1. number start with an optional sign
2. sign can be followed by nay number of digits
3. digits can be optionally followed by a decimal point, represented by a period
4. if there is a decimal point , then there should be 2 digit after decimal
5. Any Number – whether or not it has a decimal point, should be terminated by a blank.

Arunai Engineering College

An example of a state transition diagram.



State transition table

Current state	Input	Next State
1	Digit	2
1	+	2
1	-	2
2	Digit	2
2	Blank	6
2	decimal point	3
3	Digit	4
4	Digit	5
5	Blank	6

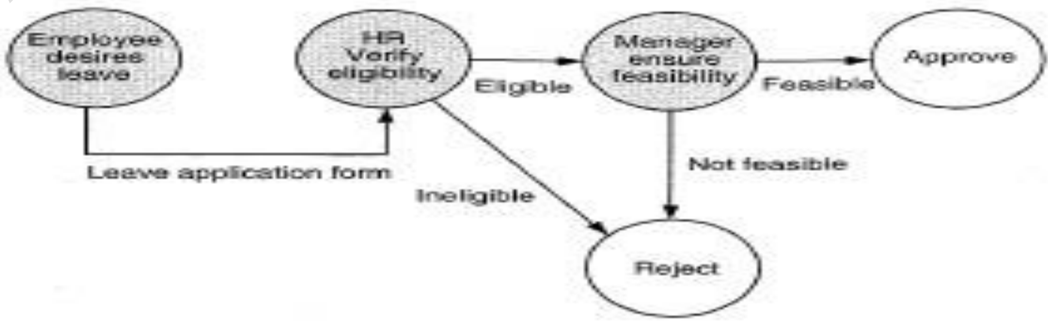
the above state transition table can be used to derive test cases to test valid and invalid numbers

1. Start the start state (state #1)
2. Choose the path that leads to the next state (ex: +/-/digit)
3. Invalid i/p in a given state, generate an error condition TC
4. Repeat the process till u reach the final state

A general outline for using state based testing methods with respect to language processors is

1. Identify the grammar for the scenario. In the above example, we have represented the diagram as a state machine. In some cases, the scenario can be a context-free grammar, which may require a more sophisticated representation of a "state diagram."
2. Design test cases corresponding to each valid state-input combination.
3. Design test cases corresponding to the most common invalid combinations of state-input.

Ex2 : Leave application by an employee (for work flow modeling)



Anna Engineering College

4) Cause effect Graphing

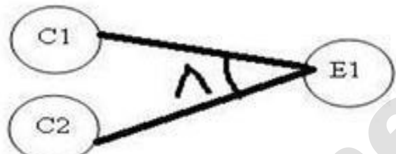
- Equivalence class partitioning does not allow testers to combine conditions .
- It is a dynamic test case writing technique.
- Cause and effect graphing is technique that can be used to combine conditions and derive an effective set of test cases that may inconsistencies in a specifications
- It restates the requirements specification in terms of logical relationship between the input and output conditions. Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.

Steps:

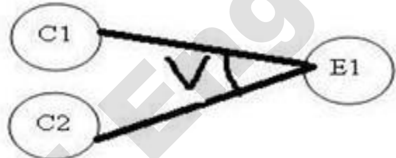
1. The tester must decompose the specification of a complex software component into lower level units
2. Identify causes and effects
   Cause - distinct i/p condition or an equivalence class of i/p conditions.
   Effect - an output condition or a system transformation
3. From the cause and effect information, a Boolean cause and Effect graph is created.
   Graph : Node → causes(Left Side) and effects (Right side). logical operators such as AND, OR and NOT and are associated with the arcs.

   Notations for constructing cause and Effect graph

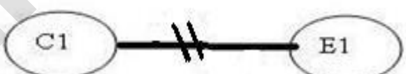
   AND – For effect E1 to be true, both the causes C1 and C2 should be true



   OR – For effect E1 to be true, either of causes C1 OR C2 should be true



   NOT – For Effect E1 to be True, Cause C1 should be false



4. The graph may be annotated with constraints that describes combinations of causes and/or effects that are not possible due to environmental or syntactic constraints
5. Convert the graph into a decision table.
6. The columns in the decision table are transformed into test cases.

Example: module that allows user to perform a search for a character in an existing string.
Step1 : decompose the specification

Input → length of the string
character to search for.

Arunai Engineering College

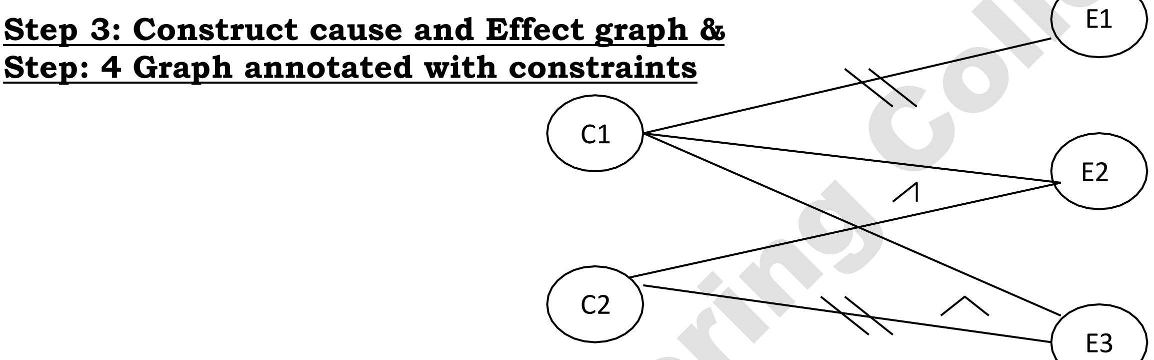
Output → Char position
NOT FOUND
out of range

Step2 : Identify <u>causes and effects</u>

C1 : Positive integer from 1 to 80
C2 : Character to search for is in String
E1 : Integer out of range
E2 : Position of character in string
E3 : Character not found.

Rules or relationship :-
If C1 and C2, then E2.
If C1 and Not C2, then E3
If not C1, then E1.

Step 3: Construct cause and Effect graph &
Step: 4 Graph annotated with constraints



Step 5: Convert the graph into a decision table(1- true , 0-false , - don't care)

	T1	T2	T3
C1	1	1	0
C2	1	0	-
E1	0	0	1
E2	1	0	0
E3	0	1	0

Step6 : Decision table are transformed into test cases

Columns are changed into testcases
Existing string → "abcde"

Test Cases	Length	Character to search for	Outputs
T1	5	C	3
T2	5	w	Not Found
T3	90		Integer out of range

5)Compatibility testing

- It ensures the working of the product with different infrastructure components (Non-functional testing).
- test case results depend on infrastructure for delivering functionality

Arunai Engineering College

- infrastructure parameter are changed , product is expected to still behave correctly and produce desired results.
- infrastructure parameter → H/W , S/W , other components
  Example
- **Test the application in same browsers but in different versions**. For e.g. to test the compatibility of site ebay.com. Download different versions of Firefox and install them one by one and test the ebay site. Ebay site should behave equally same in each version.
- **Test the application in different browsers but in different versions**. For e.g. testing of site ebay.com in different available browsers like Firefox, Safari, Chrome, Internet Explorer and Opera etc.

Parameters:

- Processor (Pentium III / IV, Xenon, SPARC)
- Architecture(32 bit / 64 bit)
- Resource Availability (RAM & Hard disk space)
- Equipment (printer , Modem, Router).
- Operating System
- Middle-tier infrastructure components (Web Server, App server)
- Back end components (Oracle, MS SQL)
- Any s/w used to generate product binaries (compiler, linker)
- Technological components (SDK, JDK)

Compatibility matrix :

Each row represents a unique combination of a specific set of values of the parameter

Ex: Mail App

Server	App Server	Web Server	Client	Browser	MS Office	Mail Server
Windows 2000 Microsoft SQL server 2000	Windows 2000 Advanced server with SP$ and .Net framework 1.1	IIS5.0	Win2K Professional	IE 6.0	Office 2k & Office XP	Exchange 5.5 & 2K
			….			

Common Techniques

- Horizontal Combination(HC): Parameters of the row grouped together for executing the test cases
- Intelligent Sampling:
  - In HC each feature of the product has to be tested with each row in the compatibility matrix→ involves time & effort
  - Various permutation and combination methods used
  - Selection of intelligent sampling based on

Arunai Engineering College

        - Information collected on the set of dependencies of the product with parameter.
        - Less dependent → removed from the list
    - Can include parameters that are part of product

Types:

- **Backward compatibility Testing** is to verify the behavior of the developed hardware/software with the older versions of the hardware/software. The product parameters required for backward compatibility is added to the compatibility matrix and are tested.
- **Forward compatibility Testing** is to verify the behavior of the developed hardware/software with the newer versions of the hardware/software.

Tools for compatibility testing:

- Adobe Browser Lab - Browser Compatibility Testing
- Secure Platform - Hardware Compatibility tool
- Virtual Desktops - Operating System Compatibility

6) User documentation testing:

User documentation testing Is done to ensure the documentation matches the product and vice versa.

User Documentation includes

Manuals	user guides	installation guides	setup guides
online help	read me files	software release notes	

Objective

- To check if what is stated in the document is available in the product
- To check if what is there in the product is explained correctly in the document.

- Product upgraded → documentation upgraded
- Lack of coordination → documentation group & testing /development group
- sitting in front of the system & verifying screen by screen , transaction by transaction , report by report
- checks language aspects (spell check & grammar)

Advantages

- Aids in highlighting problems overlooked during reviews
- High quality documentation minimizes defect reported by the customer
- Results in less difficult support calls
- New Programmer & testers can use doc. to learn the external functionality of the product
- Customer need less training & can proceed more quickly to the advanced training

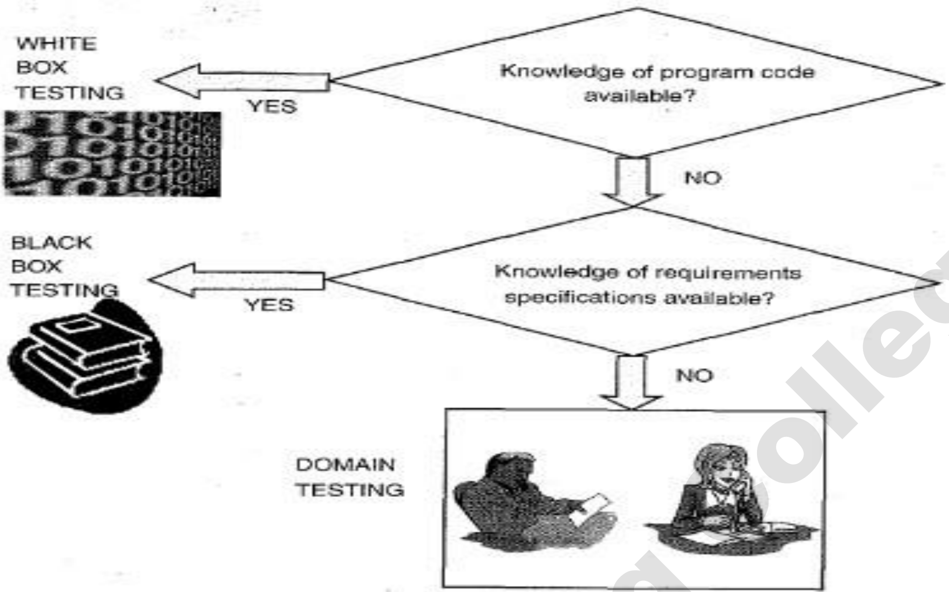
  The effort & money spent on this would form a valuable investment in the long run for the organization.

Arunai Engineering College

7) Domain Testing :

- Testing the product purely based on domain knowledge & expertise in the domain of application
- Requires business domain knowledge, Extension of black box testing

When to apply domain testing?



- Ability to design and execute test cases that relate to the people who will buy and use the software.
- Concerned about everything in the business flow
- Testing the product , not by going through the logic built into the product.
- Business flow determines the steps, not the software under test → " Business Vertical Testing"
- To Test the software for "Domain Intelligence" , tester is expected to have intelligence & knowledge of business flow
- Earlier phases of Black box Testing deals with Equivalent Class Partitioning ,Decision Table (Cause Effect Graphing)
- Domain testing is done all components are integrated and product has been tested using black box approaches.

Ex: Cash Withdrawal of ATM system

Step1 : Go to the ATM
Step2 : Put ATM card inside
Step3 : Enter Correct PIN
Step4 : Choose cash withdrawal
Step5 : Take the cash
Step6 : Exit and retrieve the card

Other Black Box testing →Required denomination is available to dispense the requested amount
Domain Testing →Whether user has got the right amount / not

Arunai Engineering College

8) Random Testing:-

If a tester randomly selects input from the domain, this is called random testing

- Eg:- if the valid input domain for a module is all positive integer between 1 and 100,
- would randomly or unsystematically select valued form within that domain; for example the values 55,24,3 might me chosen

9)Requirements Based Testing

- Deals with validating the requirements given in the SRS
- Requirements → 1) Explicit 2) Implicit
- Precondition
  - Detailed review of the requirements specification, it ensures that they are consistent , correct , complete , testable.
  - Implied requirements are converted and documented as explicit requirements → more effective
- Explicit & Implicit requirements are collected and documented as " Test Requirements specification " (TRS)
- Requirements are tracked by a Requirement Traceability Matrix (RTM)
- RTM traces all the requirements from their origin through design, development and testing.

Example : Locking (key is turned clockwise)
unlocking (key is turned anticlockwise)
Key No :123-456

Sample Requirement Specification

Sno	ReqId	Description	Priority (H, M,L)
1.	BR-01	Inserting the key numbered 123 -456 and turning it clockwise should facilitate locking	H
2.	BR-02	Inserting the key numbered 123 -456 and turning it anticlockwise should facilitate unlocking	H
3.	BR-03	only key number 123-456 can be used to lock and unlock	H
4.	BR-04	No other object can be used to lock	M
5.		....	

Requirement Traceability Matrix (RTM)

Req Id	Description	Priority (H, M,L)	Test Conditions	Test Case IDs	Phases of testing
BR-01	Inserting the key numbered 123 -456 and turning it clockwise should facilitate locking	H	Use key 123-456	Lock_001	unit Component

Arunai Engineering College

<table>
<tr><td>BR-02</td><td>Inserting the key numbered 123 -456 and turning it anticlockwise should facilitate unlocking</td><td>H</td><td>Use key 123-456</td><td>Lock_002</td><td>unit , Component</td></tr>
<tr><td rowspan="2">BR-03</td><td rowspan="2">only key number 123-456 can be used to lock and unlock</td><td rowspan="2">H</td><td>Use key 123-456 to lock</td><td>Lock_003</td><td rowspan="2">Component</td></tr>
<tr><td>Use key 123-456 to unlock</td><td>Lock_004</td></tr>
<tr><td></td><td>.....</td><td></td><td></td><td></td><td></td></tr>
</table>

- Tests for higher priority requirements will get precedence over tests for lower priority → functionality that has higher risk is tested earlier
- cross ref b/w requirements and the subsequent phases is recorded in the RTM
- RTM helps in identifying the relationship between the requirements and test cases. Combinations are
    - one (requirements) to one (Test Case)
    - one to many
    - many to one
    - many to many
    - one to none.

RTM in Requirement Based Testing:

- it is a tool to track the testing status of each requirement, without missing any requirements
- prioritization enables selecting the right features to test
- list of test cases that address the particular requirement can be viewed

Test metrics

1. Requirements addressed priority wise
2. Number of test case requirement wise
3. Total no of test cases

Test results

1. Total no of test cases passed
2. Total no of test cases failed
3. Total no of defects in requirements
4. No of requirements completed
5. No of requirements pending

Summary of Test I/P:

s.no	Req Id	Priority	Test Cases	Total test cases	Total test cases Passed	Total test cases Failed	% Pass	No of defects
1.	BR-01	H	Lock_01	1	1	0	100	1
2.	BR-02	H	Lock_02	1	1	0	100	1
3.	BR-03	H	Lock_03,04	2	1	1	50	3
4.				...				

Using above Observations can be made with respect to the requirement

Arunai Engineering College

10) Positive & Negative Testing

Positive testing

- Verifies the requirements of the product & set of expected o/p
- To prove that the product work as per specification
- is to prove that the product works as per specification and expectations. A product delivering an error when it is expected to give an error, is also a part of positive testing.
- +ve testing is done to verify the known test conditions.

Ex; Lock & Key

Req. no.	Input 1	Input 2	Current state	Expected output
BR-01	Key 123–456	Turn clockwise	Unlocked	Locked
BR-01	Key 123-456	Turn clockwise	Locked	No change
BR-02	Key 123–456	Turn anticlockwise	Unlocked	No change
BR-02	Key 123-456	Turn anticlockwise	Locked	Unlock

Negative Testing

- Negative testing is done to show that the product does not fail when an unexpected input is given. The purpose of negative testing is to try to break the system. Negative testing covers scenarios for which the product is
- not designed and coded. In other words, the input values may not have been represented in the specification of the product.
- -ve testing is done to break the product with unknowns

S. No. ✓	Input 1	Input 2	Current state	Expected output
1	Some other lock's key	Turn clockwise	Lock	Lock
2	Some other lock's key	Turn anticlockwise	Unlock	Unlock
3	Thin piece of wire	Turn anticlockwise	Unlock	Unlock
4	Hit with a stone		Lock	Lock

Difference between positive testing and negative testing

For positive testing if all documented requirements and test conditions are covered, then coverage can be considered to be 100 percent.	no end to negative testing, and 100 percent coverage –ve testing is impractical. Negative testing requires a high degree of among the testers to cover as many "unknowns" as possible to avoid at a customer site.

Summary of Black Box Testing

i/p values divided into class	1. Equivalent Class Partitioning
i/p values in range	2. Boundary Value Analysis
i/p & o/p values with multiple	3. Cause effect graphing

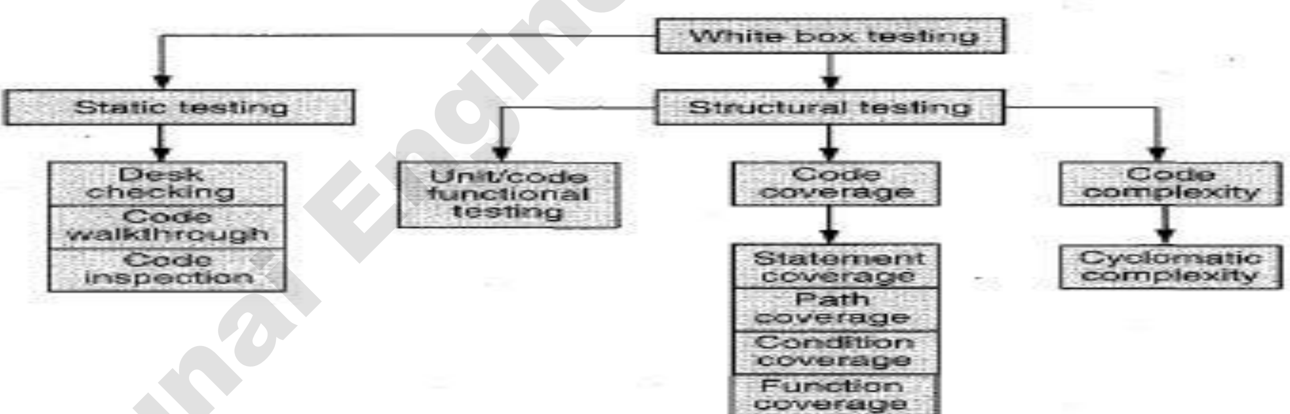
Arunai Engineering College

condition	
Checking expected & un expected i/p values	4. Positive & negative Testing
Language processor , work flow ,process flow	5. State based testing
To ensure the Requirements	6. Requirement Based Testing
To test domain expertise	7. Domain Testing
Documentation consistent with product	8. Documentation Testing

Using the White Box Approach to Test Case Design

- white box Testing → The tester's goal is to determine if all the logical and data elements in the software unit are functioning properly.
- during the detailed design phase of development - knowledge needed for the white box test design approach often becomes available to the tester in later phase of software life cycle
- White Box test design follows black box design as the test efforts for a given project progress in time

White Box	Black Box
white box based test design is most useful when testing small components.	Black box useful for both small & large s/w components
Level of detail required for test design is very high	Comparatively low



Static Vs Structural Testing

Static	Structural
Product is tested by tester by going through the source code not the executable or binaries	Tests are run by computer on the built product
Does not involve executing the program	Involves executing the program against predesigned test cases

Static test reveals :

- Code works according to the functional requirements

Arunai Engineering College

- Code has been written in accordance with the design developed earlier in the project life cycle
- Code for any functionality has been missed out
- Code handles errors properly

Static test Methods :

1. **Desk Checking of the code**
    - Informal checking done by author
    - No structured method
    - No Logs / check lists
    - Depends on knowledge of the author

<u>Disadvantages</u>

- A developer is not the best person to detect problems in his or her own code. He or she may be tunnel vision and have blind spots to certain types of problems.
- Developers generally prefer to write new code rather than do any form of testing
- This method is essentially person-dependent and informal and thus may not work consistently across all developers.

2. **Code walkthrough**
    - **Group oriented** - method and formal inspection are group-oriented methods
    - **Multiple perspective** – walkthroughs and inspections is very thin and varies from organization to organization. The advantage is that it brings multiple perspectives
    - **Multiple roles** - a set of people look at the' program code and raise questions for the author. The author explains the logic of the code, and answers the questions. If the author is unable to answer some questions, he or she then takes those questions and finds their answers

3. **Formal /Fagan Inspection:**
    1) Group oriented , highly formal & structured
    2) specific roles , requires thorough preparation

This method increases the number of defects detected by

1) demanding thorough preparation before an inspection/review;
2) enlisting multiple diverse views;
3) assigning specific roles to the multiple participants; and
4) going sequentially through the code in a structured manner.

<u>Roles :</u>

1) Author- programmer or developer
2) Moderator - expected to formally run the inspection according to the process.
3) Inspector/Reviewer - provides, review comments for the code.

Arunai Engineering College

4) Scribe - takes detailed notes during the inspection meeting and circulates them to the inspection team after the meeting.

Process:

Author & Moderator select the review team

Introductory Meeting

- o Author present his perspective
- o Typical Document (code, Design, SRS, Stds) is circulated
- o Moderator informs date ,time venue – inspection meeting

Defect Logging Meeting:

The moderator takes the team sequentially through the program code, asking each inspector if there are any defects in that part of the code. If any of the inspectors raises a defect, then the inspection team deliberates on the defect and, when agreed that there is a defect, classifies it in two dimensions

1)Major (major defects need immediate attention.**) / Minor – (**may not substantially affect a program)

2) **Systematic** (machine-specific idiosyncrasies may have to removed by changing the coding standards) / **mis execution**(happens because of an error or slip on the part of the author. example : a wrong variable in a statement)

Review Meeting (if the defect severe)

Challenges in Formal /Fagan Inspection

- Time Consuming
- Logistics & Scheduling - multiple people involved
- Not possible to review entire coding

Based on criticality & complexity of code is classified into

High ,medium	Formal Inspection
Low	Walk through , desk checking

Structural testing Methods:

The fundamental difference between structural testing and static testing is that in structural testing tests are actually run by the computer on the built product, whereas in static testing, the product is tested by humans using just the source code and not the executables or binaries.

1. Unit Functional Testing – methods fall under debugging category

a. **Initially Quick test** – the developer can perform certain obvious tests, knowing the input variables and the corresponding expected output Variables

b. **modules with Complex logic & condition** – build debug version(ex: intermediate print statement)

c. **run the product under debugger or IDE** (single stepping of instruction, break points etc)

Arunai Engineering College

2. Code Coverage Testing

Code coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing.

instrumentation

- % of code covered by testing is found by a technique
- specialized tool to rebuild the product , link with the set of lib
- Reporting on the portion of the code covered frequently, so easy to identify critical & most often code.

Types of coverage

a)Statement Coverage	b)Path Coverage
c)Condition Coverage	d)Function Coverage

a)Statement Coverage

It refers to writing test cases that execute each of the program statements. There are 4 types of programming constructs.

1. **Sequential control flow**
2. **Two-way decision statements like if then else**
3. **Multi-way decision statements like Switch**
4. **Loops like while do, repeat until and for**

1)Sequential Control flow(SC)

- Generate test data to make the program enter the sequential block, to make it go through the entire block
- this may not always be true , Asynchronous Exceptions - (for example, a divide by zero)
- Multiple Entry Point , in Non Structured Programming

SC metric= No of of statements exercised / Total No of Statements

2) Two-way decision statements –if then else

- Have data to test the then part
- Have data to test the else part
- Relevance of statement coverage ?

If the program implements wrong requirements and this wrongly implemented code is "fully tested," with 100 percent code coverage, it still is a wrong program and hence the 100 percent code coverage does not mean anything.

Ex:

```
Total =0;
If(code ==’M’)
 { Stm1;
…
   Stm7; }
Else
     Percent = value/Total *100;  /*divide be zero*/
```

Arunai Engineering College

when we test with code = "M," we will get 80 percent code coverage. But if the data distribution in the real world is such that 90 percent of the time, the value of code is not = "M," then, the program will fail 90 percent of the time (because of the divide by zero in the highlighted line).

3) multi way decision statements – switch

It can be reduced to multiple two way if statement

4) Loops – while do ,repeat until , for

Looping statements can be handled in 3 ways.

1) Skip the loop
   - so that the situation of the termination condition being true before starting the loop is tested.
2) Exercise the loop between one & max number of times
   - to check all possible "normal" operations of the loop
3) Cover the loop around the boundary (i.e n-1, n,n+1)

b)Path Coverage

statement coverage may not indicate "true coverage". path coverage gives better representation, split a program into a number distinct paths. A program can start from the beginning and take any of the paths to its completion.

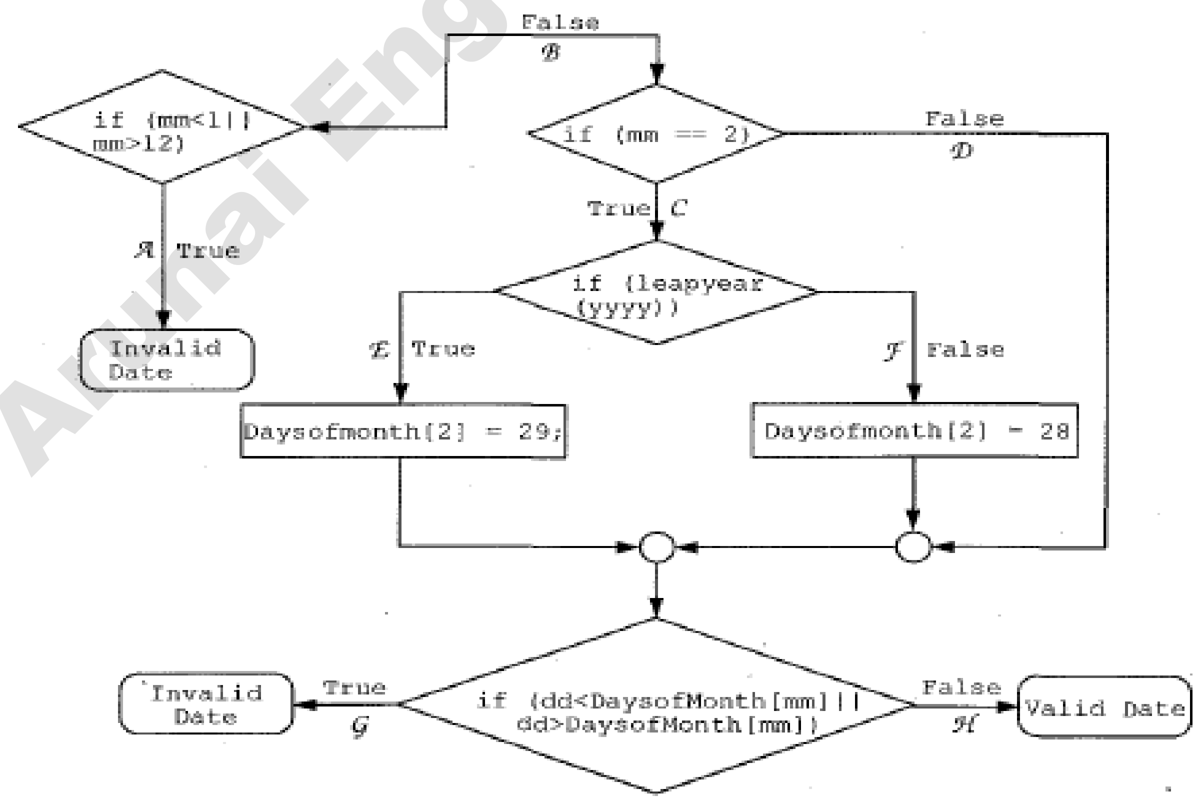
Path Coverage= No of of path exercised/Total No of of path in the program

Ex: Date validation routine . date accepted as 3 fields dd, mm, yyyy.

i/p → validate numeric i/p for date (dd , mm , yyyy)

leapyear() function (returns FALSE/ TRUE based on i/p)

array → dayofMonth[] contains No of days in each month



Anna Engineering College

The Flow chart shows different Path can be taken through program. Path label is given as A… H .

A	B-D-G B-D-H	B-C-E-G B-C-E-H	B-C-F-G B-C-F-H
If Wrong Month given	Except Feb Month with correct and wrong date given	Feb Month & Leap year with correct and wrong date given	Feb Month & Not Leap year with correct and wrong date given

Summary of Test inputs

TC ID	Path (Description)	Input	Expected o/p
TC1	A (Month Wrong Path)	20/ 0/2000	Invalid Date
TC2	B-D-G (Not Feb - days wrong)	31/4/2015	Invalid Date
TC3	B-D-H (Not Feb - days correct)	31/1/2015	valid Date
TC4	B-C-E-G (Feb , Leap Year - days wrong)	30/2/2016	Invalid Date
TC5	B-C-E-H (Feb , Leap Year - days correct)	29/2/2016	valid Date
TC6	B-C-F-G (Feb , Not Leap Year - days wrong)	29/2/2014	Invalid Date
TC7	B-C-F-H (Feb , Not Leap Year - days Correct)	10/2/2014	valid Date

C)Condition Coverage

- It is necessary to have test cases that exercise each Boolean expression and have test cases test produce the TRUE and FALSE paths.
- Further refinement of path coverage , Make sure each Boolean expression is covered for TRUE as well as FALSE paths
- Ex: Path A covered on giving mm < 1 , reporting invalid month
  Program not tested for mm> 12
- Compliers perform optimizations to minimize the number of Boolean operations and all the conditions may not get evaluated, even though the right path is chosen.
- For example, when there is an OR condition (as in the first IF statement above), once the first part of the IF (for example, mm < 1) is found to be true, the second part will not be evaluated at all as the overall value of the Boolean is TRUE.
- Similarly, when there is an AND condition in a Boolean expression, when the first condition evaluates to FALSE, the rest of the expression need not be evaluated at all.
- For all these reasons path testing is not sufficient.

Condition Coverage= No of of conditions exercised/Total No of of conditions in the program

Above formula indicates percentage of conditions covered by a set of test cases.

d)Function Coverage

- This testing finds how many functions are covered by test cases
- Ex: Database s/w -- inserting a row into the database
  Payroll app – calculate tax
- **Adv:**
  1) Functions are easier to identify
  2) Higher level of abstraction than code, easy to achieve 100%

Arunai Engineering College

3) more logical mapping to requirements than other type of coverage
4) importance of functions can be prioritized based on the importance of requirements
5) provides natural transition to black box testing

Function Coverage = No of of function exercised/Total No of of functions in the Program

3)Code Complexity Testing

While using these coverage : following questions are raised

1) which of the paths are independent ?(to minimize the test cases)
2) is there an upper bound on the number of tests to be executed to ensure all the statements have been executed at least once ?

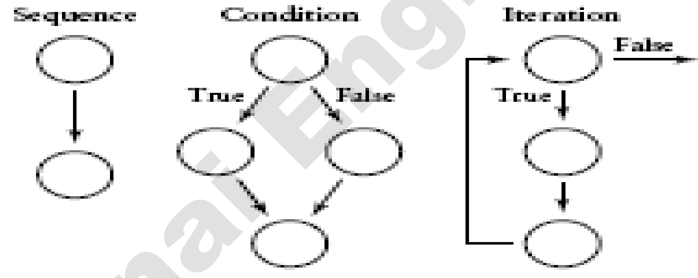
Ans : Cyclomatic complexity " a metric that quantifies the complexity of a program"

Steps in Determining Cyclomatic complexity

1) Construct Flow Graph
2) Compute cyclomatic complexity of the resultant flow graph
3) Determine a basis set of linearly independent paths
4) Prepare test cases that will force execution of each path in the basis set

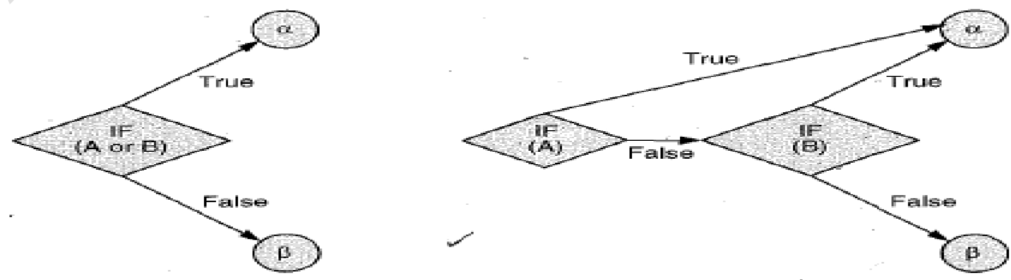
Flow Graph

- Program is represented in the form of a flow graph.
- Flow graph can be constructed like a flowchart.
- Flow graph consist of nodes and graphs.
- Representation of Programming primitives in flowgraph



Steps to convert flowchart into flow graph

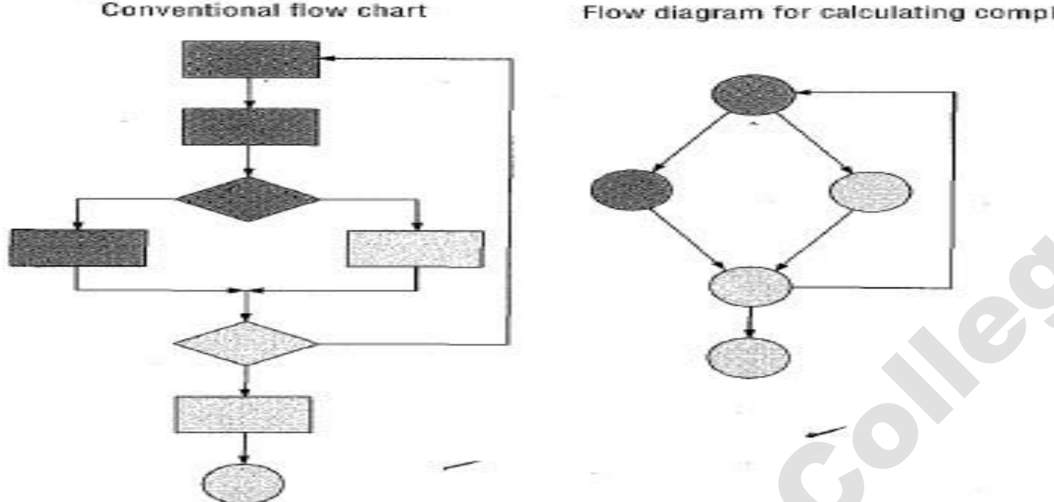
1) Identify the predicates or decision points
2) Ensure that the predicates are simple



(a) A predicate with a Boolean OR (b) An equivalent set of simple predicates

Annai Engineering College

3) Combine all sequential statements into a single node
4) When a set of sequential statements are followed by a simple predicate , combine all the sequential statements & predicate check into one node & have 2 edges emanating from this one node
5) Make sure that all the edges terminate at some node.



To Compute cyclomatic complexity : 3 ways

i) Cyclomatic Complexity V(G) = E – N + 2, where E is the number of edges and N is the number of nodes in graph G ii) V(G) = P + 1, where P is the number of predicate nodes in the flow graph G iii) V(G) =the number of regions (Closed & Outer Region)

Example : sum of all positive numbers (greater than zero)

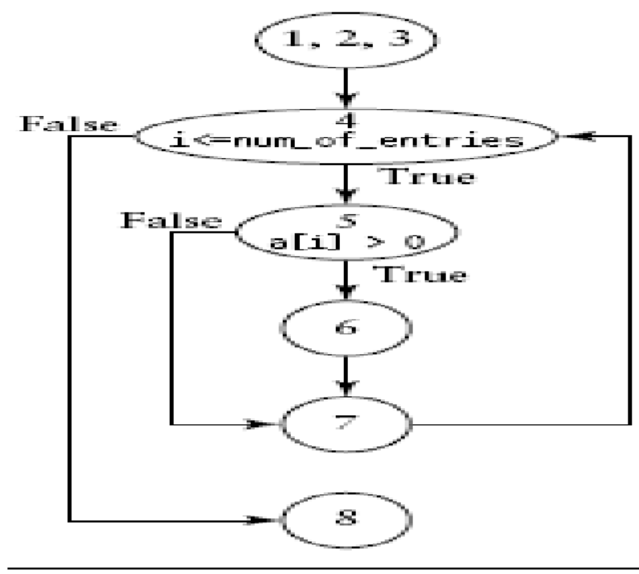
a→ array name
num_of_entries → no of elements
sum → to store total value

```
1.      pos_sum(a, num_of_entries, sum)
2.      sum=0
3.      int i=1
4.      while (i <=num_of_entries)
5.              if (a[i] >0)
6.                      sum=sum+a[i]
                endif
7.              i=i+1
        end while
8.      end pos_sum
```

Assign line no for each statement in the program before constructing the flowgraph.

Arunai Engineering College

1)Construct Flow Graph



2) Calculate the cyclomatic complexity of the resultant flow graph

i)V(G)= E - N+2	E= 7, N= 6	V(G) = 7 - 6 + 2 =3
ii)V(G)= P+1	P=2	V(G) = 2+1 = 3
iii) V(G) = No of Regions(R)	R=3	V(G) = 3

3)Determine a basis set (independent path)

A path is a sequence of control flow nodes usually beginning from the entry node of a graph through to the exit node.

(i) 1-2-3-4-8 (skip the loop)

(ii) 1-2-3-4-5-6-7-4-8 (adding number to sum)

(iii) 1-2-3-4-5-7-4-8 (not adding number to sum)

4) Prepare summary of test cases

Test case Id	Input	Expected o/p	Actual o/p	Result :Pass/Fail
TC1	num_of_entries = -5	0	0	Pass
TC2	num_of_entries = 3 30 60 20	i=1 sum=30 i=2 sum=90 i=3 sum=110 Sum=110	110	Pass
TC3	num_of_entries = 1 -30	i=1 sum=0 Sum=0	0	Pass

Arunai Engineering College

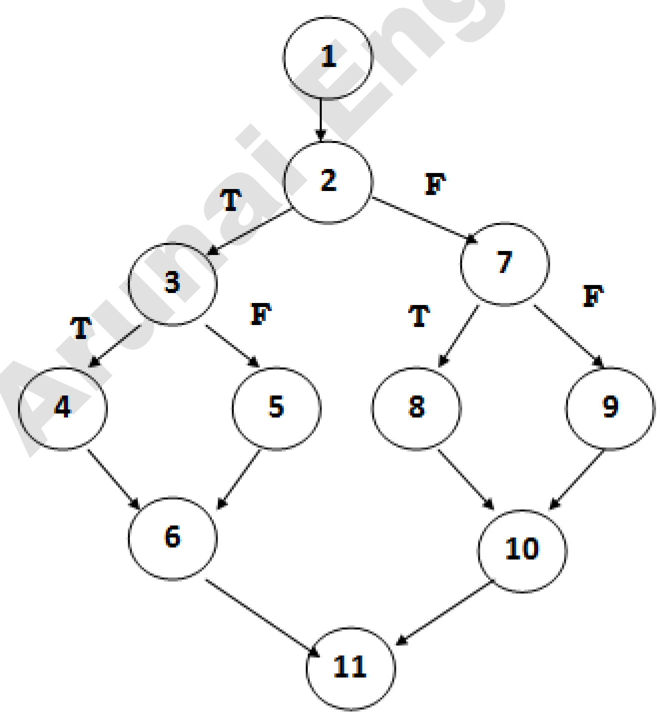
Meaning of cyclomatic complexity value

Complexity	Meaning
1-10	Well written code , testability is high , cost/effort maintain is low
10-20	Moderately complex , testability is medium ,cost/effort to maintain is medium
20-40	Very complex , testability is low ,cost/effort to maintain is high
>40	Not testable ,any amount of money /effort to maintain may not be enough

Problem: Biggest of 3 Numbers

```
1       Read A,B,C
2       If A > B then
3               If  A >C then
4                       Print " A is greater"
                Else
5                       Print "C is greater"
6               Endif
        Else
7               If  B >C then
8                       Print " B is greater"
                Else
9                       Print "C is greater"
10              Endif
11      Endif
```

1. Construct Flow Graph



Arunai Engineering College

2. Calculate Cyclomatic Complexity

1. V(G)= E - N+2   E= 7, N= 11   V(G) = 13 -11 + 2 =4
2. V(G)= P+1   P=3   V(G) = 3+1 = 4
3. V(G) = No of Regions(R)   R=4   V(G) = 4

4. Derive Basis Set

(i) 1-2-3-4-6-11 (A is greater)
(ii) 1-2-3-5-6-11 (C is greater)
(iii) 1-2-7-8-10-11 (B is greater)
(iv) 1-2-7-9-10 -11 (C is greater)

5. Summary of Test I/p

Test case Id	Input	Expected o/p	Actual o/p	Result :Pass/Fail
TC1 Path1	A= 12 B=10 C=2	A is greater	A is greater	Pass
TC2 Path2	A= 12 B=10 C= 23	C is greater	C is greater	Pass
TC3 Path3	A= 10 B=12 C= 2	B is greater	B is greater	Pass
TC4 Path4	A= 10 B=12 C= 23	C is greater	C is greater	Pass

Arunai Engineering College

Additional White Box Test Design Approaches

- Data Flow and White Box Test Design
- Mutation Testing
- Loop Testing

Test Adequacy Criteria TAC:- (stopping rule)

- Def: Tester need a framework for deciding which structural elements to select as the focus of testing, for choosing the appropriate test data and for deciding when the testing efforts are adequate enough to terminate the process with confidence that the software is working properly.
- It is minimal standards for testing a program

– Helping testers to select properties of a program to focus on during test
– Helping testers to select a test data set for a program based on the selected properties
– Supporting testers with development of quantitative objects for testing
– Indicating to testers whether or not testing can be stopped for that program

Types of TAC:

1. Program Based TAC : focus on structural properties of program , includes logic ,control structure , data flow
2. Specification based TAC: focus on program specification
3. Random TAC: ignores both program structure& specification
   Ex: TAC focus on statement/branch properties

"A test data set is statement, or branch , adequate if a test set *T* for program *P* Causes all the statements, or branches to be executed respectively"
Coverage Analysis: The TAC & the requirement that certain features of the code are to be exercised by the test case, also named as coverage criteria

Degree of coverage : when a coverage related testing goal is expressed as a percent.
degree of coverage < 100%   due to the following:-

1. The Nature of the Unit
   i. Some statements/branches may not be reachable
   ii. The unit may be simple, and not mission or safety , critical and so complete coverage is thought to be unnecessary
2. The lack of resources
   i. The time set aside for testing is not adequate to achieve 100% coverage
   ii. There are not enough trained testers to achieve complete coverage for all the units
   iii. There is a lack of tools to support complete coverage.
3. Other project related issued such as timing, scheduling and marketing constraints.
   Ex: 4 branches in s/w unit

2 are executed by planned set of test cases Degree of branch coverage : 50%	Coverage goal is not met
Develop Additional test cases & re execute the test cases	Continue until desired degree is obtained

Evaluating Test Adequacy Criteria :TAC hierarchy

- Tester can select appropriate criterion using the hierarchy
- Criteria at the top includes the Criteria at the Bottom , for example All def-use path adequacy means - tester achieved branch & statement adequacy
- Each Adequacy Criteria has both strength and weakness
- Stronger criteria → tester need more time and resource

Example : (*Sample code with data flow information)*
def → variable defined
use → variable Used
p-use → Predicate Use , variable used in condition
c-use → Computation use , variable used in calculation

Code	Data flow
1 sum=0	sum, def
2 read (n)	n, def
3 i=1	i, def
4 while (i <=n)	i, n p-use
5 read (number)	number, def
6. sum=sum+number	sum, def, sum, number, c-use
7 i=i+1	i, def, c-use
8 end while	
9 print (sum)	sum, c-use

Arunai Engineering College

Ex: DU Chain (Def-Use Path) Chain in Data flow Testing

Def-Use Path a path from a variable definition to a use is called a def-use path

Table for n

pair id	def	use
1	2	4

Table for number

pair id	def	use
1	5	6

.

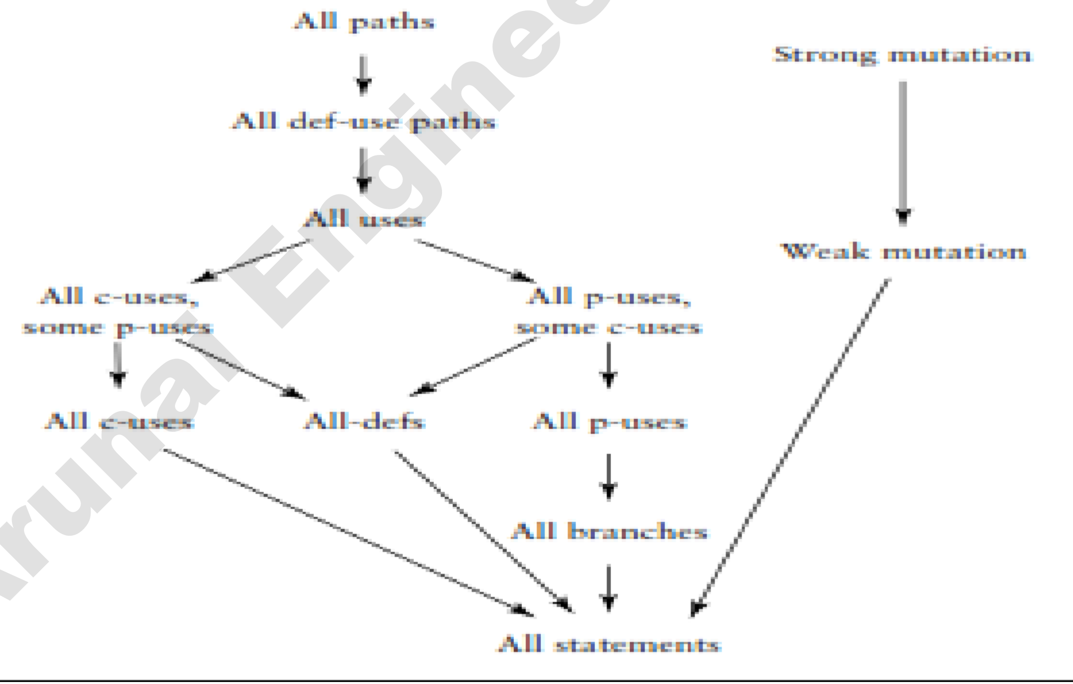
Table for sum

pair id	def	use
1	1	6
2	1	9
3	6	6
4	6	9

Table for i

pair id	def	use
1	3	4
2	3	7
3	7	7
4	7	4

Partial Ordering for Test Adequacy Criteria



Arunai Engineering College

Axioms

set of axioms that allow testers to formalize properties which should be satisfied by any good program-based test data adequacy criterion

Testers can use the axioms to

- recognize both strong and weak adequacy criteria;

- focus attention on the properties that an effective test data adequacy criterion should exhibit;
- select an appropriate criterion for the item under test;
- stimulate thought for the development of new criteria;

The axioms are based on the following set of **assumptions**
(i) programs are written in a structured programming language;
(ii) programs are SESE (single entry/single exit);
(iii) all input statements appear at the beginning of the program;
(iv) all output statements appear at the end of the program.

The **axioms/properties** described by Weyuker are the following
1. Applicability Property
2. Non exhaustive Applicability Property
3. Monotonicity Property
4. Inadequate Empty Set
5. Anti extensionality Property
6. General Multiple Change Property
7. Anti decomposition Property
8. Anti composition Property
9. Renaming Property
10. Complexity Property
11. Statement Coverage Property

Sample test data adequacy criteria and axiom satisfaction

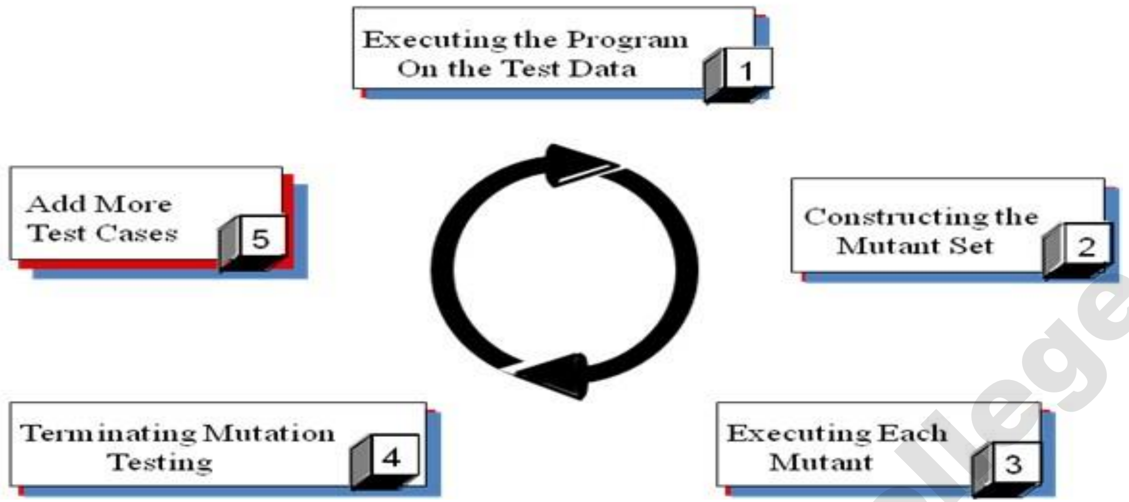
	Statement	Branch	Mutation
Axiom 1	No	No	Yes
Axiom 2	Yes	Yes	Yes
Axiom 3	Yes	Yes	Yes
Axiom 4	Yes	Yes	Yes
Axiom 5	Yes	Yes	Yes
Axiom 6	Yes	Yes	Yes
Axiom 7	No	No	Yes
Axiom 8	No	No	Yes

Mutation Testing
- is a testing technique that focuses on measuring the adequacy of test cases.
- A test case is *adequate* if it is useful in detecting faults in a program.
- A test case can be shown to be adequate by finding at least one mutant program that generates a different output than does the original program for that test case.

Arunai Engineering College

- If the original program and all mutant programs generate the same output, the test case is *inadequate*.

Basic Steps



Kinds of Mutation

A mutation is a small change in a program.

Value Mutations: these mutations involve changing the values of constants or parameters (by adding or subtracting values etc), e.g. loop bounds { being one out on the start or finish is a very common error.

Decision Mutations: this involves modifying conditions to reflect potential slips and errors in the coding of conditions in programs, e.g. a typical mutation might be replacing a > by a < in a comparison.

Statement Mutations: these might involve deleting certain lines to reflect omissions in coding or swapping the order of lines of code. There are other operations, e.g. changing operations in arithmetic expressions. A typical omission might be to omit the increment on some variable in a while loop.

Example of Testing By Decision Mutation

First test data set--M=1, N=2 , the original function returns 2

- five mutants: replace”>“ operator in if statements by (>,<,<=or=)

<table>
<tr><th>Program</th><th colspan="3">Mutants</th></tr>
<tr><td rowspan="6">function MAX(M ,N:INTEGER)
return INTEGER is
begin
if M>N then
return M;
else
return N;
end if:
end MAX;</td><td>Mutants</td><td>Outputs</td><td>Comparison</td></tr>
<tr><td>if M>=N then</td><td>2</td><td>live</td></tr>
<tr><td>if M<N then</td><td>1</td><td>dead</td></tr>
<tr><td>if M<=N then</td><td>1</td><td>dead</td></tr>
<tr><td>if M=N then</td><td>2</td><td>live</td></tr>
<tr><td>if M< >N then</td><td>1</td><td>dead</td></tr>
</table>

- Executing each mutant: adding test data M=2, N=1 will eliminate the latter live mutant, but the former live mutant remains live because it is equivalent to the original function. No test data can eliminate it.

Arunai Engineering College

ARUNAI ENGINEERING COLLEGE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
IT8076-SOFTWARE TESTING (2017 Regulation)
UNIT-III

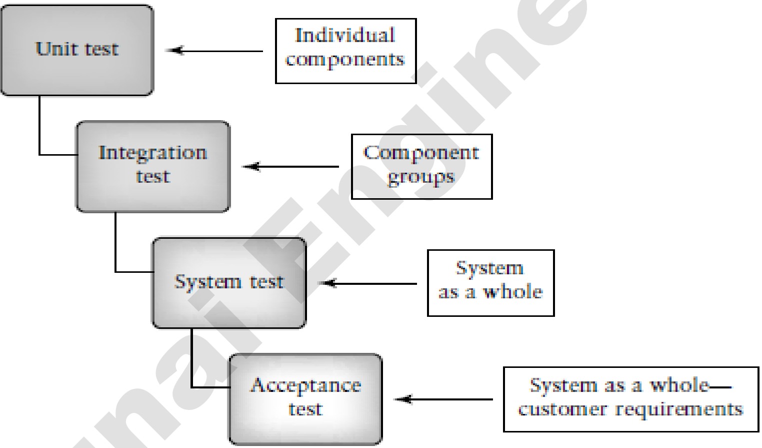
The Need of Levels of testing, Unit test , Unit test planning, Designing the unit test. Test Harness, Running the unit tests and recording results. Integration Tests- Designing Integration test- Integration Test Planning-Scenario Testing –Defect Bash Elimination-System testing -Acceptance testing-Performance Testing-Regression Testing-Internationalization Testing- Adhoc Testing-Alpha-Beta Tests-Testing OO Systems-Usability and Accessibility Testing- configuration testing – compatibility testing –testing the documentation –website testing

Arunai Engineering College

Need For Levels Of Testing:-

- Execution based software testing, especially large systems, is usually carried out at different levels
- Major Phases of testing:
    - Unit Test
    - Integration Test
    - System Test
    - Acceptance Test

Principal goal is to detect functional and structural defects in the unit. At the integration level several components are tested as group, and tester investigates component interaction. At the system level the system as a whole is tested and a principle goal is to evaluate attribute such as ability, reliability and performance



Level of Testing and Software Development Paradigms

The approach used to design and develop a software system has an impact on how a testers plan and design suitable tests.

- The major approaches to system development- 1) Bottom up 2) Top down
- These approaches are supported by two major types of programming languages-
  1) procedure Oriented and 2) Object Oriented
- The different levels of systems developed with both approached using their traditional procedural programming languages or object oriented programming languages.

- Systems developed with procedural languages
  - are generally viewed as being composed of passive data and active procedures
  - When test cases are developed the focus is on generating input data to pass to the procedures (or functions) in order to reveal defects.
- Object Oriented systems
  - are viewed as being composed of active data along with allowed operations on that data, all encapsulated within a unit similar to abstract data type.

Unit test: Functions, Procedures, Classes, and Method as Unit

- **A workable definition for a software unit is as follows**
  - A Unit is the smallest possible testable software component
    - It can be characterized in several ways. For example a unit in a typical procedure oriented software system"
      - Perform a single cohesive function
      - Can be compiled separately
      - Is a task in a work breakdown structure (from the manager's point of view)
      - Contain code that can fit on a single page or screen.
- A unit is traditionally viewed as a function or procedure implemented in a procedural (imperative) programming language.
- In object oriented systems both the method and the class/object have been suggested by researchers
- A unit may also be a small sized COTS component purchased from an outside vendor that is undergoing evaluation by the purchaser, or simple module retrieved from an in-house reuse library

Procedures and functions	Classes/objects and methods	Procedure-sized reusable components (Small-sized COTS components or components from an in-house reuse library)

Unit Test- The Need for Preparation

- The principal goal →for unit testing is insure that each individual software is functioning according to its specification
- Good testing practice → calls for unit tests that are planned and public.
- Planning includes
  - designing test to reveal defects such as functional description defects, algorithmic defects , data defects, and control logic and sequence defects.
  - Resources should be allocated and test cases should be developed, using both white and black box test design strategies.
- The unit should be tested by an independent tester (other than testers) and the test results and defects found should be recorded as apart of the unit history (made public).

Arunai Engineering College

- Each unit should also be reviewed by a team of reviewers, preferably before the unit test.
- Unit test in many cases is performed informally by the unit developer soon after the module is completed, and it compiles cleanly.
- Some developers also perform an informal review of the unit .
- To prepare for unit test the developers/ testers must perform several tasks. These are:-

1. Plan the general approach to unit testing
2. Design the test cases, and test procedures
3. Define relationships between the test
4. Prepare the auxiliary code necessary for unit test.

Unit test Planning

A general unit test plan should be prepared.It may be prepared as a component of the master test plan or a stand alone plan.It should be developed in conjunction with the master plan and the project plan for each project

- **Phase 1 : Describe Unit test Approach and Risk**

  In this phase of unit test planning the general approach to unit test planning is outlined: The test planner

1. Identifies test risks
2. Describes techniques to be used for designing the test cases for units
3. Describes techniques to be used for data validation and recording of test results
4. Describes the requirement for test harness and other software that interfaces with unit to be tested eg:- any special software needed for testing object oriented units

- During this phase the planner also identifies completeness requirements ie what will be covered by the unit test and to what degree (state , functionality, control, data flow patterns)
- Planner also identifies termination condition for unit test.
- They include coverage requirement and special cases
- Special cases may result in abnormal termination of unit test
- Planner estimate the resources needed for unit test such as hardware, software and staff and develop tentative schedule under constraints identified at that time

Phase 2:- Identify Unit Features to be Tested

- This phase requires information from the unit specification and detailed design description
- The planner determines which features of each unit will be tested, for example functions, performance requirement , state and state transition , control structures , messages and data flow patterns
- Some features will be covered by the tests, they should be mentioned and risks of not testing them be assessed.
- Input and output of each test unit should be identified.

Phase 3: Add levels of Detail to the Plan

- In this phase the planner refines the plan as produced in the previous two phases
- The planner adds new details to the approach, resource and scheduling portions of the unit test plan

Arunai Engineering College

- Eg:- Existing test cases that can be reused for this project can be identified in the phase
- Unit availability and integration scheduling information should be included in the revised version of the test plan
- Planner must be sure to include a description of how test results will be recorded.
- Test related documents that will be required for this task eg test logs, test incidents report should be described.

<u>Designing the Unit test</u>

- It is important to specify the following test design information with the unit test plan
  - The test cases (including I/O and expected output for each test cases)
  - The test procedures (steps required run the test)
  - As a part of the unit test design process, developers / tester should also describe the relationship between the tests.
  - Test suites can be defined that binds related tests together as a group.
- Test cases, test procedures and test suites may be reused from the past projects if the organization has been careful to store them so that they can be easily retrievable and reusable
- Test case design at unit level can be base on the use of black and white box design strategies
- Both of these approaches are useful for designing test cases for functions and procedures
- They are useful to designing test for individual methods in a class. This approach gives the tester the opportunity to exercise logic structure and /or data flow sequence or to use mutation analysis, all with the goal of evaluating the structural integrity of the unit

<u>Class as a testable Unit</u>

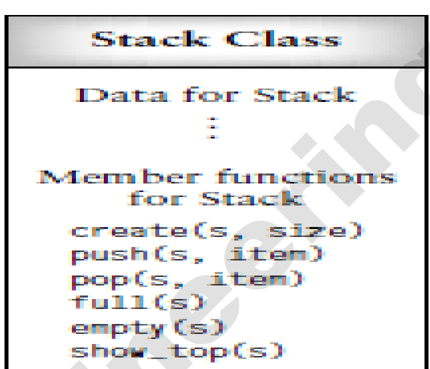
- If an organization is using the object oriented paradigm to develop software system it will need to select the component to be considered for unit test.
- Choice consist of 1) individual methods as a unit or 2) the class as a whole.
- Additional code in the form of tests harness, must be built to represent the called methods within the class. This is costly;
- Building such test harness for each individual method often require developing code equivalent to that already existing in the class itself.
- In spite of the potential advantages of testing each method individually, many developers/testers consider the class to be the component of choice for unit testing. The process of testing classes as units is sometimes called component test
- When testing on the class level we are able detect not only traditional types of defects, for example, those due to control or data flow errors, but also defects due to the nature of object oriented systems, for example, defects due to encapsulation, inheritance, and polymorphism errors.

<u>Issue 1:- Adequately Testing Classes</u>

- The potentially high costs for testing each individual method in a class

Arunai Engineering College

- These high cost will be particularly apparent when there are many methods in a class; the number can reach as high as 20 to 30.
- If the class is selected as the unit to test, it is possible to reduce these cost since many cases the methods in a single class server as drivers and stubs for one another.
- This has the effect of lowering the complexity of the test harness that needs to be developed.
- some cases driver classes that represent outside classes using the methods of the class under test will have to be developed.
- For example : create, pop, push empty, full and show_top methods associated with the stack class.
- When testers unit(or components) test this class what they will need to focus on is the operation of each of the methods in the class and the interaction between them
- For example, a test sequence for a stack that can hold three items might be:
  create(s,3), empty(s), push(s,item-1), push(s,item-2), push(s,item-3),
  full(s), show_top(s), pop(s,item), pop(s,item), pop(s,item), empty(s), . . .



Arunai Engineering College

<u>Issue 2: Observation of Object States and State Changes</u>

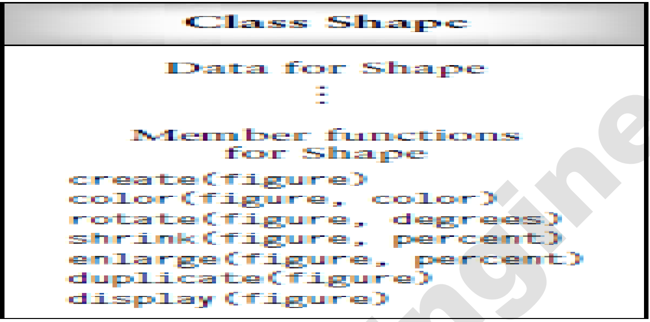
- Methods may not return a specific value to a caller
- They may instead change the state of an object .
- The state of an object is represented by a specific set of values for its attributes or state variables
- Methods often modify the state of an object, and the tester must ensure that each state transistor is proper
- The test designers can prepare a state table that specifies states , the object can assume, and then in the table indicate sequence of messages and parameters that will cause the object to ensure each state.
- When the test are run the tester can enter results in this table. The first call to the method *push* in the stack class, changes the state of the stack so that empty is no longer true. It also changes the value of the stack pointer variable, top.
- To determine if the method *push* is working properly the value of the variable *top* must be visible both before and after the invocation of this method. In this case *show_top* within the class may be called to perform this task.
- The methods full and empty also probe the state of the stack. A sample augmented sequence of calls to check the value of top and the *full/ empty* state of the three item stack is :

Issue 3:- The Retesting of classes-I

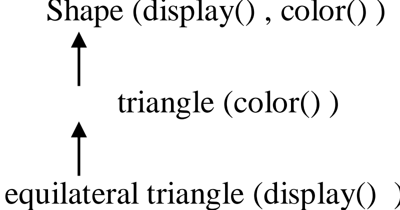
- One of the most beneficial features of object oriented development is encapsulation → used to hide information
- A program unit , in this case a class, can be built with a well defined public interface that proclaims its services to client classes. The implementation of the services is private. Client who use the service s are unaware of implementation details. The interface is unchanged , making changes to the implementation should not affect the client classes. A tester object oriented code would therefore conclude that only the class with implementation changes to its methods needed to be retested.
- In an object-oriented system, if a developer changes a class implementation that class needs to be retested as well as all the classes that depend on it. If a superclass, for example, is changed, then it is necessary to retest all of its subclasses

Issue 4:- The Retesting of classes-II

- Classes are usually a part of a class hierarchy where there are existing inheritance relationships
- Subclasses inherit methods from their super classes
- Tester may assume that once a method in a super class has been tested , it does not need retested in a subclasses that inherit it.



- There may be overriding of methods where a subclass may replace an inherited methods with a locally define methods.
- Designing a new set of test cases may be necessary.
- This is because the two methods may be structurally different



- Suppose the shape superclass has a subclass, triangle, and triangle has a subclass, equilateral triangle. Also suppose that the method display in shape needs to call the method color for its operation.

Arunai Engineering College

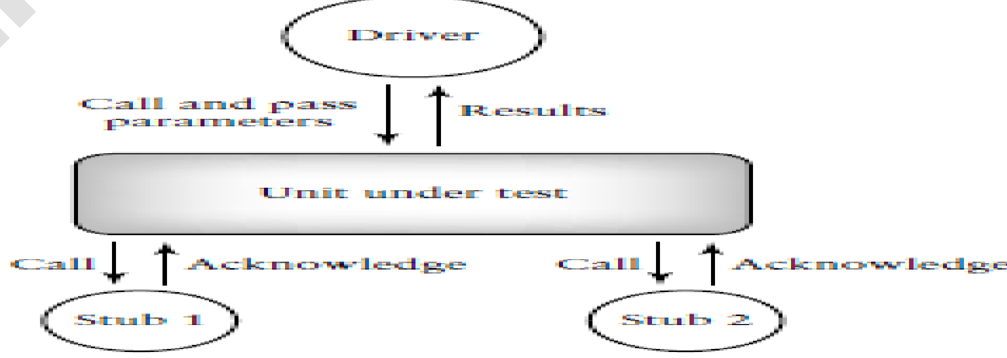
- Equilateral triangle could have a local definition for the method display. That method could in turn use a local definition for color which has been defined in triangle.
- This local definition of the color method in triangle has been tested to work with the inherited display method in shape, but not with the locally defined display in equilateral triangle.
- This is a new context that must be retested. A set of new test cases should be developed.
- The tester must carefully examine all the relationships between members of a class to detect such occurrences.

The Test Harness

- The auxiliary code developed to support testing of units and components is called as test harness
- The harness consist
  - **drivers → call the target code**
  - **stubs → represent modules it calls.**
- The development of drivers and stubs requires testing resources.
- The drivers and stubs must be tested themselves to insure they are working properly and that they are reusable for subsequent releases of the software
- Drivers and stubs can be developed at several levels of functionality
- Eg:- a driver could have the following options and combinations of options:
  - Call the target unit
  - Do 1, and pad pass input parameters from the table
  - Do 1,2, and display parameters
  - Do 1,2,3 and display result (output parameters)

The stub should also exhibit bit different levels of functionality

- For example a stub could
  - Display a message that it has been called the target unit
  - Do1, and display any input parameters passes from the target units
  - Do1,2, and pass back result from a table
  - Do1,2,3 and display result from table



Arunai Engineering College

Running a Unit tests and Recording Results

- Unit test can begin when
  - The unit become available from the developers
  - The test cases have been designed and reviewed
  - The test harness and any other supplemental to supporting tools
  - The status of the test efforts for a unit, and a summary of test results must be recorded in a unit test worksheet

Unit Test Worksheet

Unit Name: ____________________

Unit Identifier: ____________________

Tester: ____________________

Date: ____________________

Test case ID	Status (run/not run)	Summary of results	Pass/fail

- It is very important that the tester at any level of testing to carefully record, review and check test results.
- The tester must determine from the results whether the unit has passed or failed the test
- If the test is failed, the nature of the problem should be recorded in what is sometimes called the <u>test incident report.</u>
- Differences from expected behavior should be described. When a unit fails a test there may be several reasons for the failure.
  - fault in the unit implementation
  - A fault in the test case specification (the input or the output was not specified correctly)
  - A fault in test procedures execution(the test should be rerun)
  - A fault in the test environment (perhaps a database was not set up properly)
  - A fault in the unit design (the code correctly adheres to the design specification , but the latter is incorrect)
- When a unit has been completely tested and finally passes all of the required tests it is ready for integration

Integration Test-Goals

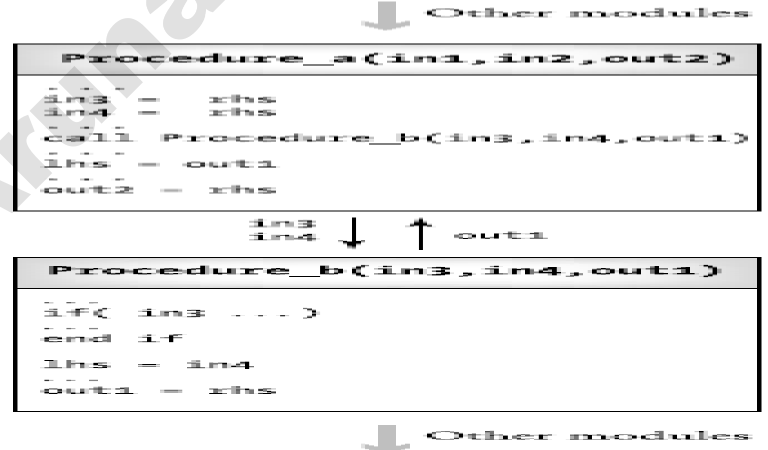
- Integration test for procedural code has two major goals
  - To detect defects that occur on the interfaces of units
  - To assemble the individual unit into working subsystem and finally a complete system that is ready for system test.
- In unit test the tester attempts to detect defects that are related to the functionality and structure of the unit.
- Some simple unit interfaces are more adequately tested during integration test when each unit is finally connected to a full and working implementation of those unit it calls, and those that call it.

Arunai Engineering College

- Few minor expectations, integration test should only be performed on unit that have been reviewed and have successfully passed unit testing.
- A tester might believe erroneously that since a unit has already been tested during a unit test with a drivers and stubs, it does not need to be retested in combinations with other units during integration test.
- Integration testing works best as an iterative process procedural oriented system.
- One unit at a time integrated into a set of previously integrated modules which have passed a set of integration tests.
- The interface and functionality of the new unit is combination with the previously integrated units is tested
- When a subsystem is built from units integrated in the stepwise manner, then performance , security and stress test can be performed in this subsystem.
- Integrating one unit at a time helps tester in several ways.
- It keeps the number of new interfaces to be examined small, so that can focus on these interfaces only.
- Experienced tester know that many defects occur at module interface.
- Another advantage is that the massive failures that often occur multiple units are integrated at once is avoided.
- Approach also helps the developers, it allows defect search and repair confined to a small known number of components and interfaces
- Integration process is object oriented systems is driven by assembly of the classes into cooperating groups.
- The cooperating groups of classes are tested as a whole and then combined into higher level groups.

Designing Integration tests

- Integration test → using a black or white box approach , Some unit test can be reused
- Since many error occur at module interfaces, test designers need to focus on exercising all input/output parameter pairs and all calling relationships
- The tester needs to insure the parameters are of the correct type and in the correct order.
- The author has had the personal experience of spending many hours trying to locate a fault that was due to an correct ordering of parameters in the calling routine



Arunai Engineering College

- Example : Procedure_b is being integrated with Procedure_a. Procedure_a calls Procedure_b with two input parameters in3, in4. Procedure_b uses those parameters and then returns a value for the output parameter out1. Terms such as lhs and rhs could be any variable or expression.
- The parameter could be involved in a number of *def* and/or *use* data flow patterns
- The actual usage patterns of the parameters must be checked at integration time.
- Some black box test used for module integration may be reusable from unit testing.
- When units are integrated and subsystems are to be tested as a whole, new tests will a have to be designed to cover the functionality tests at the integration level are the requirements document and the user manual.
- Tester need to work with requirement analyst to insure that the requirement are testable, accurate and complete.
- Black Box tests should be developed to insure proper functionally and ability to handle subsystem stress.
- Integration Testing of clusters of classes also involves building test harness which in this case are special classes of objects built for testing
- Class testing we evaluated intra class method interaction , at the cluster level we test inter class method interaction as well
- We want to insure that message are being passed properly to interfacing objects, object state transition are correct when specific events occur , and that the cluster are performing their required functions.
- A group of cooperative classes is selected for a test as a cluster.(packages in java)
- If developers have used the Coad and Yourdon's approach , then a subject layer could be used to represent a cluster.

Integration test Planning

- Integration test must be planned
- Planning can begin when high level design is complete so that the system architecture is defined.
- Documents relevant to integration test planning are the requirement document, the use manual and usage scenario. These document contains structure charts, the state charts and data dictionary , cross reference table , module interface description
- The strategy for integration of the unit must be defined .
- For procedural-oriented system
  - the order of integration of the units of the units should be defined. This depends on the strategy selected. Consider the fact that the testing objectives are to assemble components into subsystems and to demonstrate that the subsystem functions properly with the integration test cases.
- For object-oriented
  - systems a working definition of a cluster or similar construct must be described, and relevant test cases must be specified. In addition, testing resources and schedules for integration should be included in the test plan. The plan includes the following items:
    - Cluster this cluster is dependent on

Arunai Engineering College

- A natural language description of the functionality of the cluster to be retested.
- List a classes in the cluster
- A set of cluster test cases

Integration Testing Types:-

Integration testing can be viewed as

1) type of testing
2) phase of testing.

Integration is defined to be a set of interactions, all defined interaction among the components need to be tested. The architecture and design can give the details of interactions within the systems, however testing the interactions between one system and another system required detailed understanding of how they work together.

Integration Testing As a Type of Testing :-

Integration testing means testing of interfaces. They are

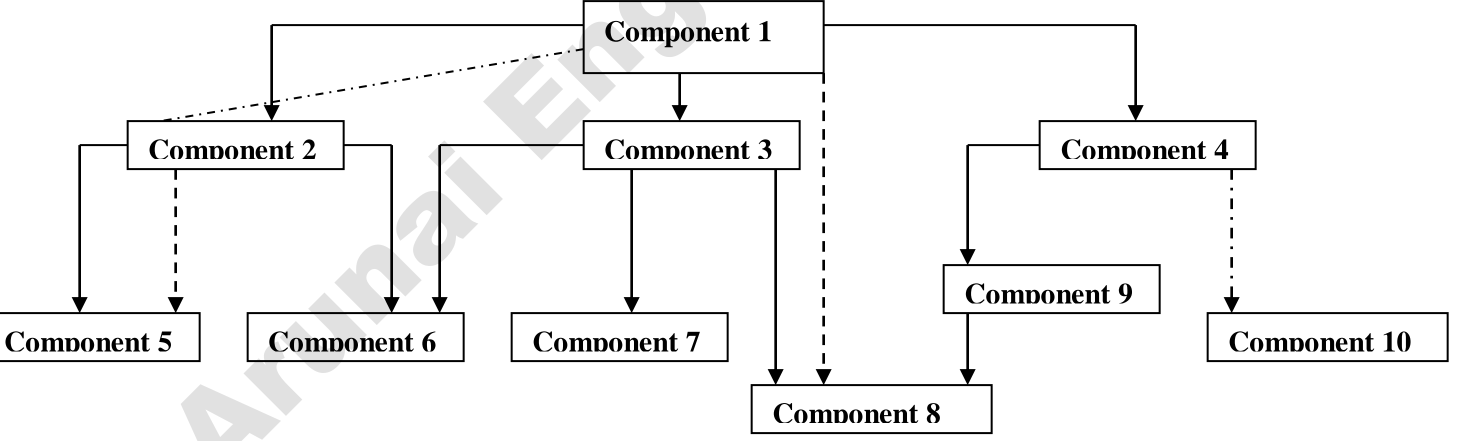
Internal Interfaces - provide communication across two modules within a projects or product, internal to the product, and not exposed to the customer or external developers

Exported or ***External Interfaces***.. - Exported interfaces are those that are visible outside the product to third party developers and solution providers.

"Intergration Testing Type Focuses on testing interfaces that are "Implicit and Explicit" and "Internal and External"

Implicit interface -> Documentation given

Explicit interface-> No documentation given



"A set of Modules and Interfaces"

In the above diagram, it is clear that there are at least 12 interfaces between the modules to be tested (9 explicit and 3 explicit). Now what will be the order of interface to be tested. There are several methodologies available , to in decide the order for integration testing. These are as follows:-

1. Top Down Integration
2. Bottom up Integration

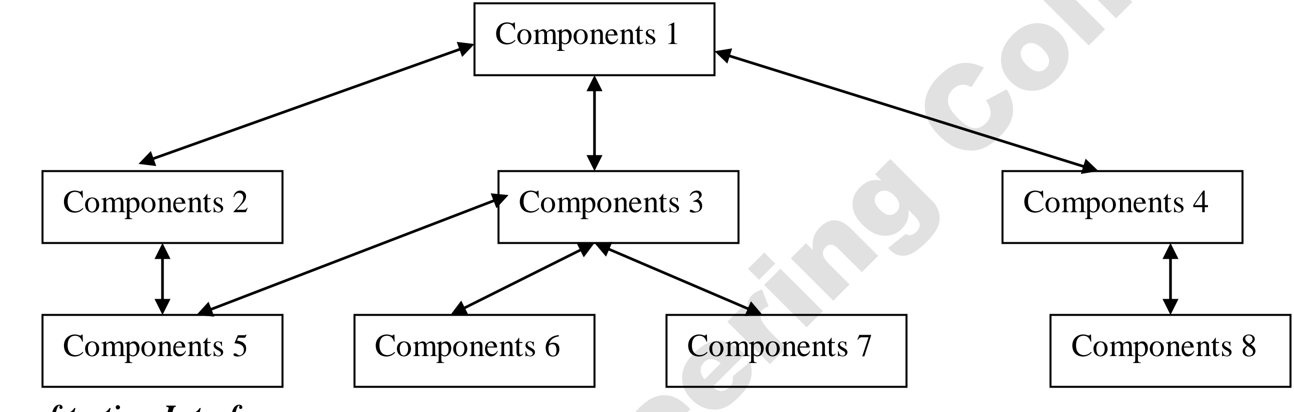
Arunai Engineering College

3. Bi-Directional Integration
4. System (Big Bang) Integration

Top-Down Integration :-

Integration Testing involves testing the topmost component interface with other components in same order as you navigate from top to bottom, till we cover all the components.

To understand this methodology, we will assume that a new product/ software development where components become available one after another in the order of component number specified .The integration starts with testing the interface between Component 1 and Component 2 .To complete the integration testing all interfaces mentioned covering all the arrows, have to be tested together. The order in which the interfaces are to be tested is depicted in the table below. In an incremental product development, where one or two components gets added to the product in each increment, the integration testing methodology pertains to only those new interfaces that are added .



Order of testing Interfaces

Steps	Interfaces Tested
1	**1-2**
2	**1-3**
3	**1-4**
4	**1-2-5**
5	**1-3-6**
6	**1-3-6-(3-7)**
7	**(1-2-5)-(1-3-6-(3-7))**
8	**1-4-8**
9	**(1-2-5)-(1-3-6-(3-7))-(1-4-8)**

For example , assume one component (component 8) is added for the current release , then the integration testing for current release need to include steps 4,7,8 and 9.

To optimize no of steps in integration(optimization of elapsed time) , following steps can be combined ,

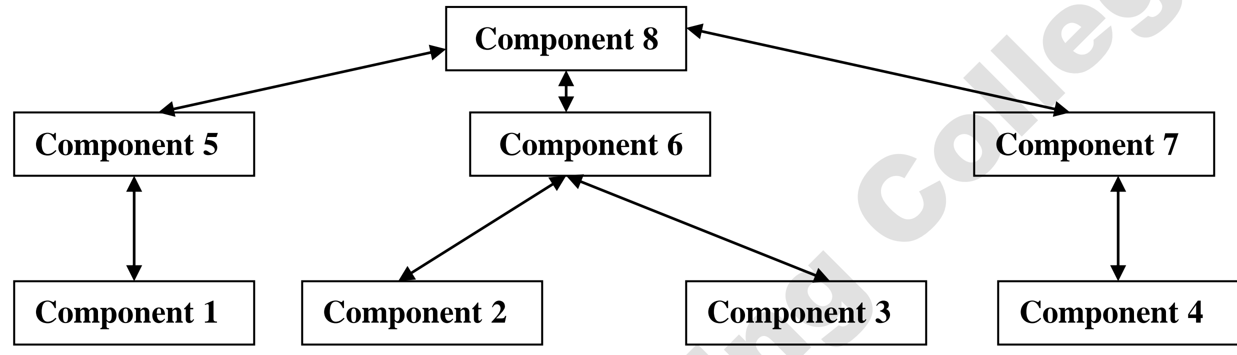
- step 6,step 7 → executed as single step,
- step 8,step 9

Arunai Engineering College

Subsystem : set of components and their related interfaces can deliver functionality without expecting the presence of components is called as sub system . Ex: components in steps 4, 6 and 8 can be considered as subsystem.

Bottom-Up Integration:-

Bottom-up integration is just the opposite of top-down integration, where the components for a new product development becomes available in reverse order, starting from the bottom. Testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers. Logic Flow is from top to bottom and integration path is from bottom to top. Navigation in bottom-up integration starts from component 1 converting all sub systems , till components 8 is reached. The order is listed below. The number of steps in the bottom up can be optimized into four steps. By combining step2 and step3 and by combining step 5-8 in the previous table.



Order of Interface tested using Bottom Up Integration

Steps	Interfaces Tested
1	1-5
2	2-6,3-6
3	2-6-(3-6)
4	4-7
5	1-5-8
6	2-6-(3-6)-8
7	4-7-8
8	(1-5-8)-(2-6-(3-6)-8)-(4-7-8)

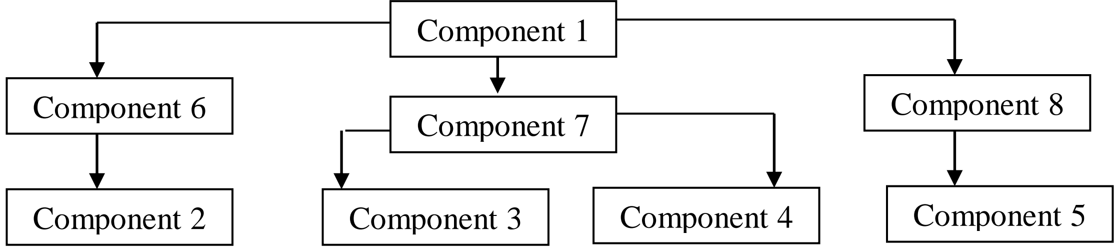
Bidirectional Integration:- (*Sandwich Integration*) Bi directional integration is a combination of the top-down and bottom –up integration approaches used together to derive integration steps. Let us assume software components become available in the order mentioned by the component numbers.

The Individual component 1, 2, 3, 4, and 5 are tested separately and bi-directional integration is performed initially with the use of studs and drivers. Drivers are used to provide upstream connectivity while stubs are provided for downstream connectivity.

A driver is a function which redirects the request to some other components and stubs simulate the behavior of a missing components. After the functionality of these integrated components are tested, the drivers and stubs are

Arunai Engineering College

discarded .Once component 6,7 and 8 becomes available, the integration methodology then focuses only on those components , as these are the components which need focus and are new.



Steps for Integration Using Sandwich Testing :-

Steps	Integration Tested
1	**6-2**
2	**7-3-4**
3	**8-5**
4	**(1-6-2)-(1-7-3-4)-(1-8-5)**

System (Big Bang) Integration:-

System Integration means that all the components of the system are integrated and tested as a single unit. Integration testing, which is testing of interface, can be divided into two types:-

- Components or Sub-System Integration
- Final Integration testing or system Integration

Big bang Integration is deal for a product where the interfaces are stable with less number of defects.

There are some major important disadvantages that can have a bearing on the release dates and quality of a product are as follows :-

1. When a Failure or defects is encountered during system integration, it is very difficult to locate the problem, to find out in which interface the defects exists. The debug cycle may involve focusing on specific interfaces and testing them again.
2. The ownership for correcting the root cause of the defects may be a difficult issue to pin point.
3. When integration testing happens in the end , the pressure from the approaching release date is very high. This pressure on the engineers may cause them to compromise on the quality of the product .
4. A certain components may take an excessive amount of time to be ready. This precludes testing other interfaces and wastes time till the end.

Choosing Integration Methods:-

Sno	Factors	Suggested Integration Methods
1	**Clear Requirement and Design**	**Top Down**
2	**Dynamically, Changing Requirements, Design, Architecture**	**Bottom-Up**

Arunai Engineering College

3	**Changing Architecture, Stable Design**	**Bi-Directional**
4	**Limited Changes to existing Architecture with less Impact**	**Big Bang**
5	**Combination of all the above**	**Select one of the above after careful analysis**

<u>Integration Testing As a Phase of testing :-</u>

"All testing activities that are conducted from the point where two components are integrated to the point where all system components work together , are considered a part of the integration testing phase."

The Integration testing phases focuses on finding defects which predominantly arise because of combining various components for testing, and should not be focused on for component or few components .Integration testing as a type focuses on testing the interfaces. This is a subnet of the integration testing phase.

<u>Scenario Testing:-</u>

Scenario testing is defined as a "set of realistic user activities that are used for evaluating the products" .It is also defined as testing involving customer scenarios. There are two methods to evolve scenario

1. System Scenario
2. Use case Scenario/ Role Based Scenarios.

<u>System Scenario:-</u>

System Scenario is a method where by the set of activities used for scenario testing covers several components in the system. The following approaches can be used to develop system scenarios.

Story-line : Develop a story-line that combines various activities of the product that may be executed by an end user.

Life-cycle / state transitions: Consider an object, derive the different transitions / modification that happen to the object and derive scenarios to cover them . Ex: Savings Bank Account(opening , deposit , withdraw , interest calculation etc) → applied to money object

Deployment / implementation details from customer: develop a scenario from a known customer deployment / implementation details and create set of activities by various users in the implementation

Business verticals: Visualizing how a product / software will be applied to different business verticals and create a set of activities as scenarios (e.g., purchasing function is done differently in pharmaceuticals, software houses , government organization →so make the product multi purpose)

Battle-ground scenarios: Create some scenarios to justify "the product works" and some scenarios to "try and break the system" to justify "the product doesn't work."

The set of scenarios developed will be more effective if the majority of the approaches mentioned above are used in combination, not in isolation. Scenario should not be set of disjointed activities which have no relation to each other. Any activity in a scenario is always a continuation of the previous activities. Effective Scenarios will have combination of current customers implementation foreseeing future use of product, and developing

Arunai Engineering College

adhoc test cases. Coverage is always a big question with respect to functionality in scenario testing. This testing is not meant to cover different permutations and combinations of features and usage in a product .

Coverage of Activities by Scenario Testing

End User Activity	Frequency	Priority	Applicable Environments	No of times Covered
1.Login to Application	High	High	W2000,W2003,XP	10
2. Create an Object	High	Medium	W2000,XP	7
3.Modify Parameters	Medium	Medium	W2000,XP	5
4.List Object Parameters	Low	Medium	W2000,XP,W2003	3
5.Compose Mail	Medium	Medium	W2000,XP	6
6.Attach Files	Low	Low	W2000,XP	2
7.Send Composed Mail	High	High	W2000,XP	10

Use Case Scenarios:-

A use case Scenario is a stepwise procedure on how a user intends to use a system, with different user roles and associated parameters. A use case scenario can include stories, pictures and deployment details. Use cases are useful for explaining customer problems and how the software can solve those problems without any ambiguity

Example:-

The scenario above is explaining a example of withdrawing a cash from a bank. A customer fills up a cheque and gives it to an official in the bank. The official verifies the balance in the account from the computer and gives the required cash to the customer .The customer in this example is a actor, the clerk the agent , and the response given by the computer which gives the balance in the account , is called the system response

Actor	System Response
User would like to withdraw cash and inserts the card in ATM machine	Request for Password or PIN (Personal identification number)
User Fill-in the Password or PIN	
	Validate the password or PIN
	Give a list containing types of accounts
User selects an account type	
	Ask the user for amount
User Fill-in the amount of cash required	
	Check availability of funds Update account balance Prepare receipt Dispense cash
Retrieve cash from ATM	
	Print receipt

Actor and System Response in Use Case for ATM cash withdrawal

Annai Engineering College

DEFECT BASH:-

1. Defect bash is an *ad hoc* testing, done by people performing different roles to bring out all types of defects. It is very popular among applications development companies, where the products can be used by people who perform different roles. The testing by all the participants during the defect bash is not based on written test cases. Defect bash brings together plenty of good practices that are popular in testing industry. They are as Follows :-Enabling people to "***cross boundaries and test beyond assigned area***"
2. Bringing different people performing different roles together in the organization for testing - "***Testing isn't for testers alone***"
3. Let everyone in organization use the product before delivery - "***Eat your own dog food***"
4. Bringing fresh pairs of eyes to uncover new defects – "***Fresh eyes have less bias***"
5. Bringing in people who have different levels of product understanding, to test the product together randomly – "***Users of software are not the same***"
6. Testing doesn't wait for the time taken for documentation – "***Does testing wait till all documentation is done?***"
7. Enabling people to say the "system works" as well as enabling them to "break the system" – "***Testing isn't to conclude that the system works or doesn't work***"

Even though it is said that defect bash is an ad hoc testing, not all activities of defects bash are un planned. All the activities in the defect bash are planned activities, except for what to be tested .It involves several

steps:-

1. **Choosing the frequency and duration of defect bash.**
2. **Selecting the right product build.**
3. **Communicating the objectives of each defect bash to everyone**
4. **Setting up and monitoring the lab for defect bash.**
5. **Taking action and fixing issues.**
6. **Optimizing the effort involved in defect bash.**

1. Choosing frequency and duration

- Too frequent or too few rounds may not meet objective
- Optimize duration involved

2. Selecting right product build

- Good-quality product
- Regression tested build
- Too many defects spoil confidence

3. Communication objective of defect bash

- Purpose & objective has to be clear

Arunai Engineering College

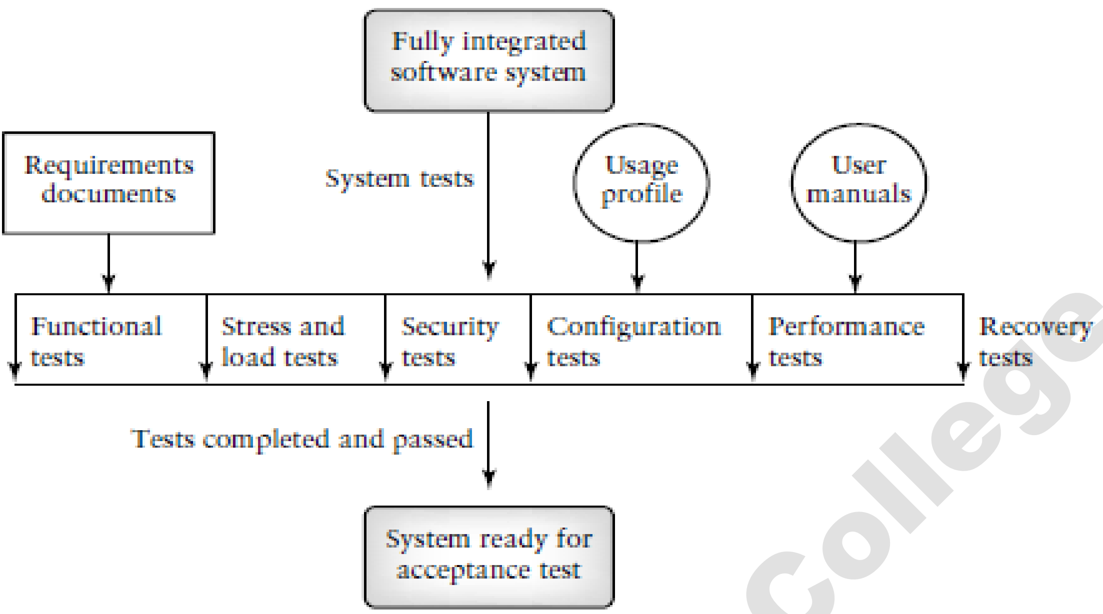
- Areas of focus to be communicated
- Defects that can be found easily by test team shouldn't be objective

4. **Setting up and monitoring lab**
    - Right configuration and resources
    - Easy install & set-up help
    - Optimized for both functional & non-functional defects
    - Monitor all resources (RAM, disk, CPU, network)
5. **Taking actions and fixing issues**
    - Duplicate defects
    - Not possible to look at each defect alone due to volume
    - Code reviews and inspections
    - Communication to all users on defects and their resolution

<u>**System Testing**</u>

- When integration tests are completed, a software system has been assembled and its major subsystem have been tested
- System test planning should begin at the requirement based (black box) test
- System test planning is a complicated task. There are many components of the plan that need to be prepared such as test approaches, costs, schedules , test cases and test procedures
- System testing itself requires large amount of resources
- The goal→to ensure that the system performs according to its requirements.
- System test evaluates both functional behavior and quality requirement such as reliability, usability, performance and security.
- The phase of testing is especially useful for detecting external hardware and software interface defects. Eg:- those causing race conditions, deadlocks , problems with interrupts and exception handling.
- The organization will want to sure that the quality of the software has been measured and evaluated before users/client are invited to use the system.
- In fact system test serves as a good <u>rehearsal scenario</u> for acceptance test.
- System test often requires many resources, special lab equipment.
- The best scenario is for the team to be part of an independent testing group.
- There are several types of system tests
    - Functional testing , Performance Testing
    - Stress testing , Configuration testing
    - Security Testing ,Recovery testing
    - reliability and usability testing.

Arunai Engineering College



- A load → is a serious of input that stimulated a group of transaction

  A transaction is a unit of work seen from the system user's view

  A transaction consist of set of operations that may be performed by a person , software system or a device that is outside the system.

  A use case can be used to describe a transaction
- Ex : a telecomm system → load that simulated a series of phone calls (transactions) of particular types and lengths arriving from different locations
- A load can be a real load, that is we can put the system under test to real usage by having actual telephone users connected to it.
- Loads can also produced by tools called load generators , they will generate test input data from system test. Load generators can be simple tools that outputs a fixed set of predetermined transaction

Functional testing

- Functional test at system level are used to ensure that the behavior of the system adheres to the requirements specification. All functional requirements for the system must be achievable by the system.
- Example→personal finance system is required to allow users to set up account, add, modify and delete entries in the accounts, and print reports, the function based system and acceptance test must ensure that the system can perform these tasks
- Functional test are black box in nature ,The focus is on the inputs and proper output for each function
- Improper and illegal inputs must also be handled by the system
- Goals
  - All types or classes of legal input must be accepted by the software

Arunai Engineering College

- All classes of illegal inputs must be rejected (however the system should remain available)
- All possible classes of system output must exercised and examined
- All effective system states and state transition must be exercised and examined
- All functions must be exercised

- If a failure is observed, a formal test incident report should be completed and returned with the test log to the developer for code repair. Managers keep track of these forms and reports for quality assurance purposes, and to track the progress of the testing process.

Performance Testing

There are two major requirements:

- **Functional Requirement**:-

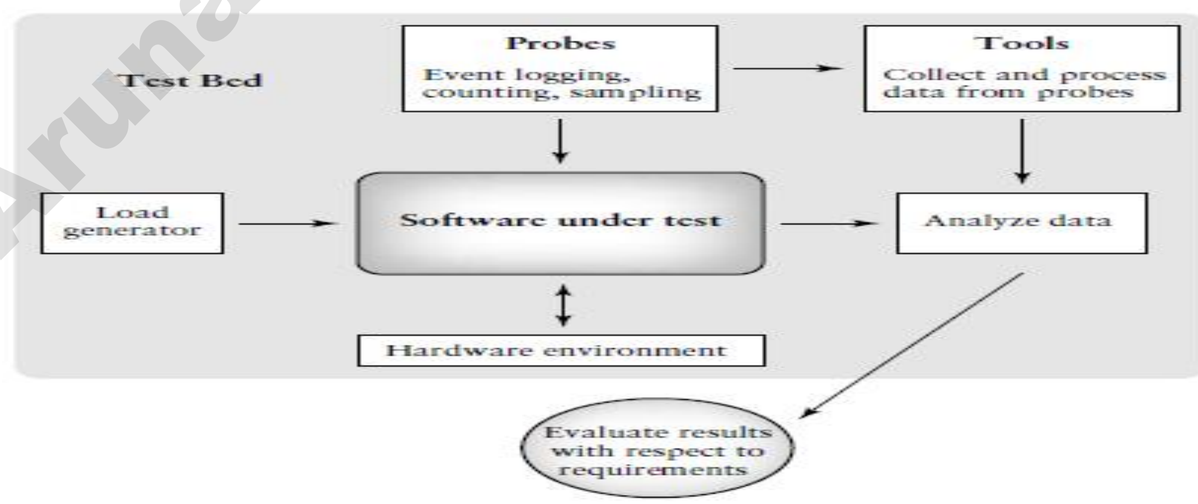
Users describes what function the software should perform. We test for compliance of these requirements at the system level with the functional based system test.

- **Quality Requirement** :-

There are nonfunctional in nature but describes quality levels expected for the software. One example of a quality requirement is performance level, the users may have objectives for the software system in terms of memory use , response time , throughput and delay

Goal : to see if the software meets performance requirements

- Testers also learn from performance test whether there are any hardware or software factors that impact on the systems requirement .
- Performance testing allows the testers to tune the system, ie to optimize the allocation of system resource
- Performance objectives stated clearly → requirement documents , system test plans
- Results of Performance system test is quantifiable ex:no of CPU cycles , response time , no of transactions per second.
- Resources for the performance testing must be allocated in the system test plan, Example of resources are given below in a diagram



Arunai Engineering College

- A source of transaction→ to drive the experiment ,For example if you were performance testing an operating system you need a stream of data that represent typical user interactions .Typically the source of transaction for many system is load generator .
- An experimental test bed → that includes hardware and software the system under test interacts with. The test bed requirement sometimes includes special laboratory equipment and space that must be reserved for the tests.
- Instruments or probes→ that help to collect the performance data, probes may be hardware or software in nature. Some probe tasks are event counting and event duration measurement. Eg:- if you are investigating memory requirements for your software you could use a hardware probe that collected information on memory usage as the system executes. The tester must keep in mind that the probes themselves may have an impact on system performance
- A set of tools to collect, store, process and interpret the data. Very often , large volume of data are collected, and without tools the testers may have difficulty in processing and analyzing the data in order to evaluate true performance levels.

Stress Testing

- When a system is tested with a load that causes it to allocate its resources in maximum amounts, this is called stress testing
- Eg:-if an OS is required to handle a 10 interrupts / second and the load cause 20 interrupt/ second, the system is being stressed
- Goal → try to break the system; find the circumstance under which it will crash, this is sometimes called "*breaking the system*"
- Stress testing is important →because it can reveal defects in real time and other types of systems, as well as weak areas where poor design could cause unavailability of services.
- Stress testing often uncovers →race conditions, deadlocks , depletion of resource in unusual or un planned patterns , and upset in normal operation of the software system.
- System limits and threshold values are exercised , Hardware and software interactions are stretched to the limit
- Stress testing is supported by many of the resource used for performance test as shown in previous diagram , This includes the load generator , The tester set the load generator parameter so that load levels cause stress to the system
- Stress testing is important from the user/client point of view
- When system operate correctly under conditions of stress then client have confidence that the software can perform as required.

Configuration Testing

- It allows developer/tester to evaluate system performance and availability when hardware exchanges and reconfigurations occurs.
- Software Systems interact with hardware devices such as disc drivers, tape drivers and printers

Arunai Engineering College

- Many Software system also interact with multiple CPU some of which are redundant.
    - Eg:- a printer of type X should be substitutable for a printer of type Y, CPU A should be removable from a system composed of several other CPUs
    - Sensor A should be replaced with Sensor B
- Software will have a set of commands or menus that allow users to make these configuration changes
- If a system does not have specific requirements for device configuration changes then large-scale configuration testing is not essential.

- According to Beizer configuration testing has the following objectives
    1. Show that all configuration changing commands and menus work properly
    2. Show that all interchangeable and that they each enter the proper states for the specified conditions
    3. Shows that the system performance level is maintained when devices are interchanged, or when they fail
- Several types of operations should be performed during configuration test, some sample operations for tester are:-
    - Rotate and Per mutate the position of devices to ensure physiological/logical device permutations work for each device
    - Induce malfunctions in each devices , to see if the system properly handles the malfunction
    - Induce multiple device malfunctions to see how the system reacts
    - These operation will help to reveal problems (defects) relating to hardware and software when hardware exchange, and the reconfiguration occur.

Security Testing:-

- Designing and testing software system insure that they are safe and secure is a big issue facing software developers and test specialist
- Safety and Security is a big issue→ because of the Internet
- Users/Client should be encouraged to make sure their security needs are clearly known at requirement time, so that security issues can be addressed by designers and Testers.
- Computer Software and data can be compromised by
    - Criminals, intent on doing damages, stealing data and information, causing denial of service , invading privacy
    - Errors on the part of honest developers/ maintainers who modify, destroy or compromise data because of misinformation , misunderstanding , and/or lack of knowledge

Attacks can be random or systematic. Damage can be done through various means such as:-

- Viruses
- Trojan Horses
- Trap Doors
- Illicit channels

Arunai Engineering College

The effect of security breaches could be extensive and can cause

- Loss of information
- Corruption of information
- Privacy violations
- Denial of service

- Physical, psychological and economic harm to process or property can result from security breaches
- Developers try to ensure the security of their systems through use of protection mechanism such as passwords, encryption , virus checkers and the detection and elimination of trap doors
- Password checking and example of other areas to focus on during security testing are described below
    - **Password Checking**:- Test the password checker to insure that users will select a password that meets the condition descried in the password checker specification. Equivalence class partitioning and boundary value analysis based on the rules and conditions that specify a valid password can be used to design the tests .
    - **Legal and Illegal Entry with password**:- Test for legal and illegal system/data access via legal and illegal passwords.
    - **Password Expiration:-** If it is decided that password will expire after certain time period, tests should be designed to insure the expiration period is properly supported and that users can enter a new and appropriate password.
    - **Encryption:-**Design test cases to evaluate the correctness of both encryption and decryption algorithm for systems where data/message are encoded
    - **Browsing:-** Evaluate browsing privileges to insure that unauthorized browsing doesn't occur. Tester should attempt to browse illegally and observe system responses. They should determine what types of private information can be inferred by both legal and illegal browsing
    - **Trap Doors:-** Identify any unprotected entries into the system that may allow access through unexpected channel (trap doors) .Design test cases that attempt to gain illegal entry and observe results. tester will need to support of designer and developers for this task
    - **Viruses:-** Design test to insure that system virus checkers prevent or curtail entry of viruses into the system. Tester may attempt to infect the system with various viruses and observer the system response.

Recovery Testing:-

- Recovery testing subjects a system to losses of resources in order to determine if it can recover properly from these losses. This type of testing is important for transaction system.
- Eg:- on line banking software
- A test scenario might be to emulate loss of device during a transaction, Test would determine if the system could return to a well known state ,and that no transaction have been compromised.
- System with automated recovery are deigned for this purpose

Arunai Engineering College

- They usually have multiple CPU and /or multiple instance of devices , and mechanism to detect the failure of the device, They are also called as "*CHECK POINTS*"
- Beizer advises that tester focus on the following areas during recovery testing
- ***Restart:-*** The current system state and transaction state are discarded The most recent checkpoint record retrieved and the system initialized to the state in the checkpoint record. Tester must insure that all transaction have been reconstructed correctly and that all devices are in proper state. The system should then be able to begin to process new transaction
- ***Switchover:-*** The ability of the system to switch to a new processor must be tested .Switch over is the result of a command or detection of faulty processor by a monitor
- All transaction and processes must be carefully examined to detect:-
  - Loss of transaction
  - Merging of transaction
  - Incorrect Transactions
  - An unnecessary duplication of transaction

Difference between functional and non functional Testing

System test contains both functional and non functional Testing

Testing aspect	Functional Testing	Non Functional Testing
Involves	Product Features and functionality	Quality Factor
Tests	Product behavior	Behavior & Experience
Result Conclusion	Simple steps written to check expected results	Huge data collected and analyzed
Results Varies Due to	Product Implementation	Product Implementation , resources and configuration
Testing Focus	Defect detection	Qualification of product
Knowledge required	Product and domain	Product ,domain, design ,architecture ,statistical skills
Failures normally due to	Code	architecture , design ,code
Testing Phase	Unit, component, integration , system	System
Test case Repeatability	Repeated Many Times	Repeated only in case of failures and for different configuration
Configuration	One time setup for a set of test cases	Configuration changes for each test case
Example	1. Design / architecture verification 2. Business vertical testing 3. Deployment Testing 4. Beta Testing 5. Certification standards and Testing for compliance	1. Scalability Test 2. Performance Test 3. Reliability Test 4. Stress Test

Arunai Engineering College

ACCEPTANCE TESTING

- It is done by the customer or by the rep of the customer to check whether the product is ready for use in the real life environment.
- Customer defines a set of test cases that will be executed to qualify and accept the product
- Small in numbers, black box type of test cases
- Written to execute real life scenarios , verifying both functional & non functional aspects of the system
- Done prior to product delivery , sometimes jointly developed by the customer and product organization

1. **Acceptance Criteria**

   a. **Acceptance Criteria(AC) – Product acceptance**

   Acceptance criteria is not meant for executing test cases that have not been executed before. hence existing testcases are looked at and certain categories of test cases can be grouped as AC.

   Ex: all performance TC should pass to meet response time requirements

   b. **Acceptance Criteria – Procedure acceptance**

   It can be defined based on the procedures followed for delivery. It could be documentation and release media. Example

   - User , admin and troubleshooting doc should be part of the release
   - Along with binary code , source code of the product build scripts to be delivered in CD
   - A minimum of 20 employees are trained on the product usage prior to deployment

   c. **Acceptance Criteria – service level agreements(SLA)**

   service level agreements are part of contract signed by the customer and product organization. Important contract items are taken and verified.

   For Ex: time limits to resolve defects mentioned in SLA

   i. All major defects that come up during first 3 months of deployment need to be fixed free of cost
   ii. Down time of the implemented system should be less than 0.1%
   iii. All major defects are to be fixed within 48 hours of reporting

2. **Selecting test cases for Acceptance testing**

   a. End to End functionality verification
   b. Domain Test
   c. User Scenario test
   d. Basic Sanity Test
   e. New Functionality Test
   f. A few Non functional Tests
   g. Test Pertaining to legal obligations and service level agreements
   h. Acceptance test data

3. **Executing Acceptance Tests**

   - Acceptance Tests done by either 1)product organization 2) customer

Arunai Engineering College

- If it is done by product organization ,forming a team is an important activity.
- It contains → Product management , support, consulting team
    - 90 % --people with business process knowledge , 10% -- tech testing team
- Testing Team may or may not aware of testing , so appropriate training on the product and the process must be given . This could be in-house training material.
- The testing team members constantly interact with acceptance team members & help them
    - to get required test data,
    - select and identify test cases
    - analyze the acceptance test result
- During test execution , the acceptance test team reports its progress regularly

REGRESSION TESTING

- Regression testing is not a level of testing, but it is the retesting of software that occur when changes are made to ensure that new version of the software has retained the capability of the old version and no new defects has been introduced due to the changes. Regression Testing can occur at any level of test
- Ex:- when unit test are run the unit may pass a number of these tests until one of the test does reveal a defect. The unit is repaired and then retested with all the old test cases to ensure that the changes have not affected its functionality
- Regression testing is **selective re-testing** of the system with an objective to ensure that the bug fixes work and those bug fixes have not caused any un-intended effects in the system
- This testing is done to ensure that:
    - The bug-fixes work
    - The bug-fixes do not create any side-effects

Regression Testing – Types

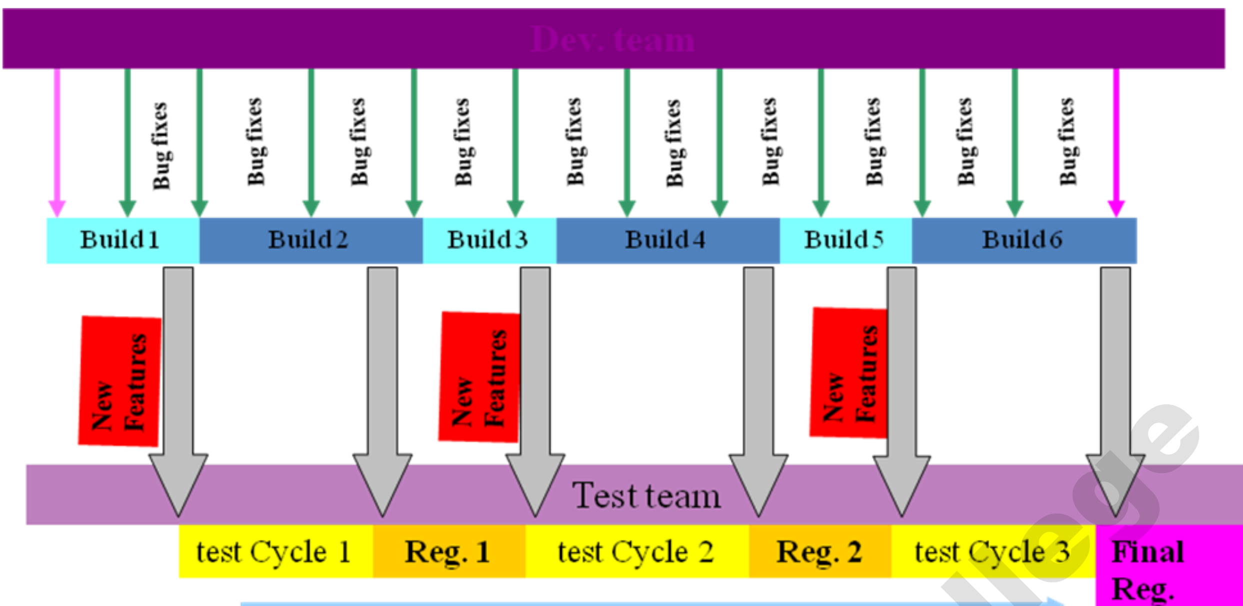
I. Final regression testing

- Unchanged build exercised for the minimum period of "cook time" (gold master build)
- To ensure that "**the same build of the product that was tested reaches the customer**"
- More critical than any other type of testing
- Used to get a comfort feeling on the product prior to release

II. Regression testing

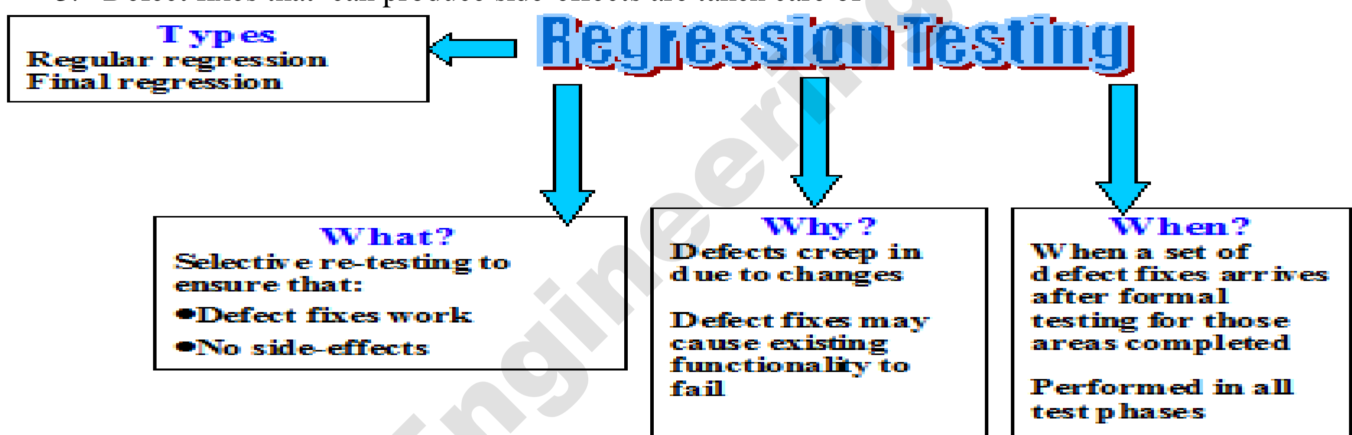
- To validate the product builds between test cycles
- Unchanged build is recommended but not mandatory
- Used to get a comfort feeling on the bug fixes, and to carry on with next cycle of testing
- Also used for making intermediate releases (Beta,Alpha)

Arunai Engineering College



When to do regression testing?

1. A reasonable amount of initial testing is already carried out
2. A good no of defects have been fixed
3. Defect fixes that can produce side-effects are taken care of



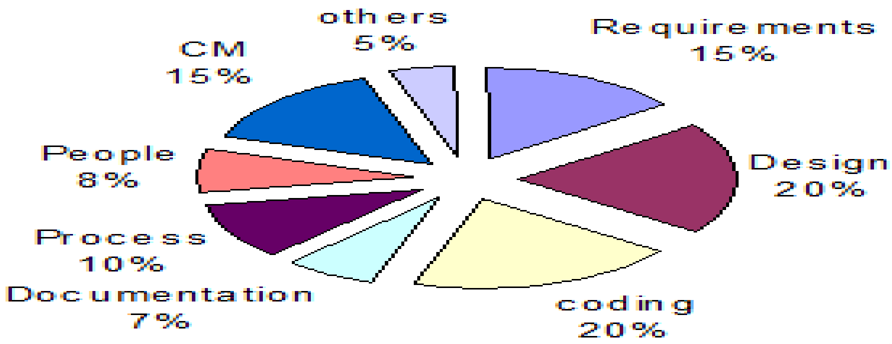
STEPS IN REGRESSION TESTING

1. Performing initial smoke tests
2. Understand the criteria for Selecting test cases
3. Classifying test cases
4. Methodology for selecting the TC
5. Resetting test cases for execution
6. How to conclude results

1. Performing initial smoke tests

- Identify the basic functionality that product must satisfy
- Designing the test cases to ensure that these functionality works , package them into smoke test suite
- Ensuring that every time the product is build this suite is run successfully before anything else is run

Arunai Engineering College

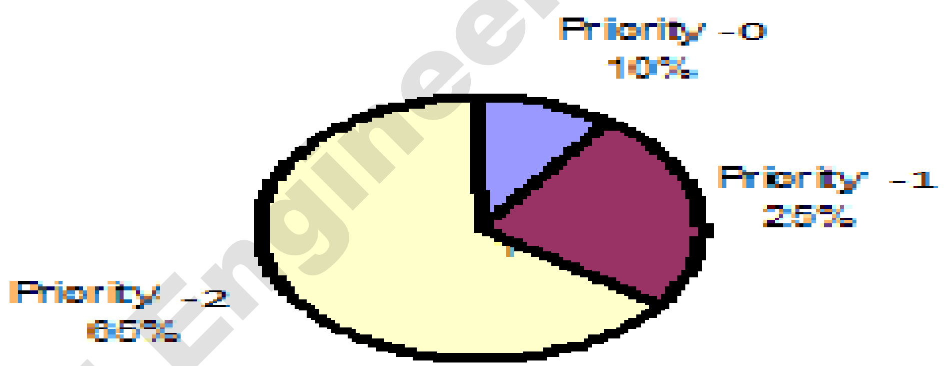


- If this suite fails , escalating to the developer to identify the changes or roll back to the state where smoke test suite succeeds

2. Understand the criteria for Selecting test cases

1. Include TC that has max defects.
2. Include TC where changes are made.
3. Include TC that test the basic functionality.
4. Include TC in which problems are reported.
5. Include TC that test end-to-end behavior.
6. Include TC for positive conditions.
7. Include the TC that are visible to the user.

3. Classifying test cases



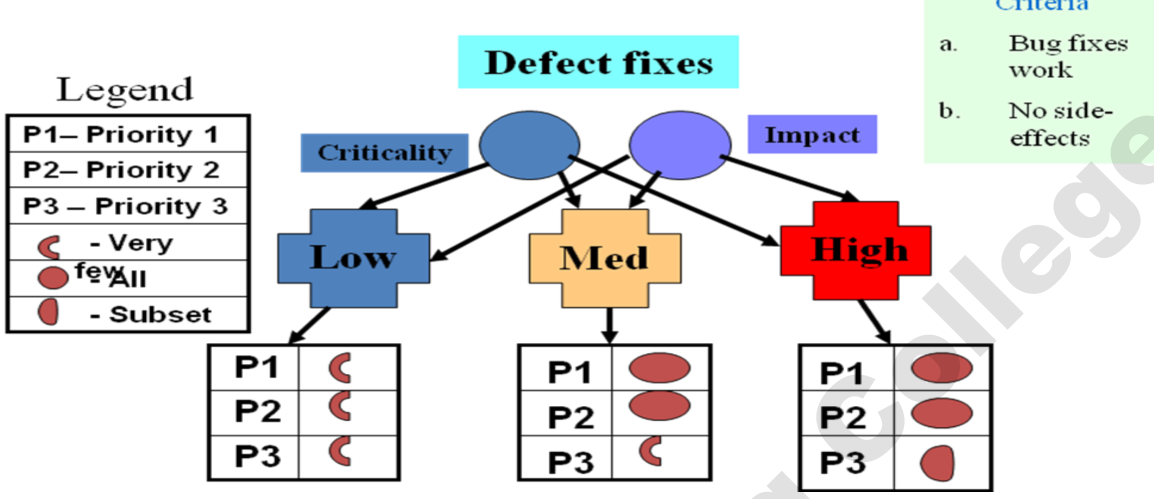
Priority -0	Called sanity Test case Check basic functionality & are run for accepting the build for further testing Done when a project goes through major change
Priority -1	Uses the basic and normal setup and these test cases deliver high project value to both development team and to customer
Priority -2	It deliver moderate project value Executed as a part of testing cycle

4. Methodology for selecting the TC

	Criticality & Impact of	Action

Arunai Engineering College

	defect fixes	
Case 1	Low	Select few Test cases from the test case database(TCDB), execute them , they fall under priority 0,1,2
Case 2	Medium	execute test cases from priority 0,1 , few from Priority 2
Case 3	high	Execute all test case from priority 0,1 & carefully select sub set of TC from Priority 2



Alternative methodology :

1. Regress all
2. Priority based (priority 0,1, 2)
3. Regress Changes
4. Random Regression
5. Context based dynamic regression

5.Resetting test cases for execution

- When there is a major change in the product
- When there is a change in the build procedure that affects the product
- In a large release cycle where some test cases have not been executed for a long time
- When you are in the final regression test cycle with a few selected test cases
- In a situation in which the expected results could be quite different from history

6. How to conclude results

Current Result from regression	Previous Results	Conclusion	Remarks
FAIL	PASS	FAIL	• Regression failed and • Apply RESET guidelines and proceed after getting new build
PASS	FAIL	PASS	• Bug fixes are working and • Continue your regression to find side effects

Arunai Engineering College

FAIL	FAIL	FAIL	• Bug fixes not working or not provided or • Wrong selection
PASS(with work around)	FAIL	Analyze the work round and if satisfied mark as PASS	• Work round needs good review as they create side effects
PASS	PASS	PASS	• This test case could have been included for finding side-effects or Wrong selection

Arunai Engineering College

INTERNATIONALIZATION TESTING:-

- **Introduction**
- **Primer**
- **Terminology**
- **Test phases for I18n**
- **Enabling testing**
- **Locale testing**
- **$I_{18}n$ validation**
- **Fake language testing**
- **Language testing**
- **Localization testing**
- **Tools**

Introduction:-

Market of software is becoming truly global. The advent of Internet has removed some of the technology barriers on widespread usage of software products and has simplified the distribution of Software Products

Building Software for the International market, supporting multiple languages, in a cost effective and timely manner is a matter of using internationalization standards throughout the software development life cycle- from requirements capture through design, development, testing and maintenance.

If some Guidelines are not followed in the SDLC for internationalization, the effort and additional cost to support every new language will increase significantly overtime. Testing for Internationalization is done to ensure that the software does not assume any specific language or conventions associated with a specific language. Testing for language or conventions associated with a specific language. Testing for Internationalization has to be done in various phases of SDLC.

Terminology Used in Internationalization

Definition of Language :-

- Language – Language is a tool used for communication. Language has Semantics or the meanings associated with the sentences. For the same language , the spoken usage , word usage and grammar could vary from country to another, however the character /alphabets may remain the same in most cases.

Character Set

- ASCII – American Standard Code for Information Interchange:
  - Uses 8 bit for representing characters

- Basic ASCII uses 7 bits (128 chars) and extended ASCII uses 8 bits (256 chars)
  - European characters and punctuation symbols are easily represented in extended ASCII
  - It also includes accented chars (ñ, á, é, í, ó, ú)

- Double Byte Character Set (DBCS) :
  - Many of the languages (Chinese & japanese) can be represented in 8 bits.
  - DBCS uses 16 bits to represent characters.
  - In DBCS, 65536 different characters can be represented

- Unicode:
  - ASCII & DBCS represents characters of a single language
  - Unicode represents all characters of all languages
  - Unicode assigns a unique code to each character no matter what language or program or platform
  - Uses 16 bit encoding
  - Unicode transformation format : Specifies algorithmic mapping of character into Unicode
  - Each language has a unique number in Unicode

Microsoft operating System	Path to Internationalization
Windows 3.1	ANSI
Windows 95	ANSI and limited Unicode
Windows NT 3.1	First OS based on Unicode
Windows NT 4.0	Display of Unicode characters
Windows NT 5.0	Display and input of Unicode characters

- Locale
  - Each of the languages is spoken differently in different countries and states
  - There could be many countries speaking the same language, using the same characters, punctuations, etc.
  - But some conventions may be different (currency and date format)
  - For example, English is used widely in the US and India, but
    - Currency : $ and Rs.
    - $1,000,000 and Rs. 1,00,000
  - There could be multiple currencies in a country (Euro and Franc in France)
  - There could be multiple locale for a language in the same country

Terminology

- **Internationalization**($I_{18}n$)
  - also called $I_{18}n$, the subscript 18 is used to mean that there are 18 characters between "I" and the last "n" in the word Internationalization

Arunai Engineering College

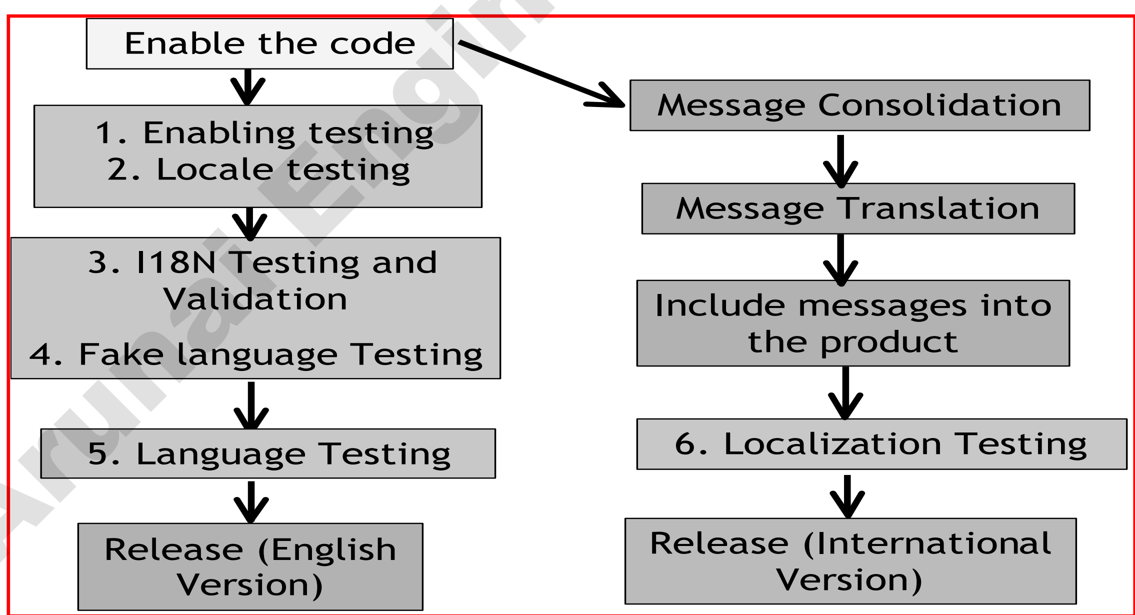
- Testing is done in various phases to ensure that all those activities are done right is called Internationalization testing or $I_{18}n$ testing.
- Represents all activities to make products available to the international market
- Includes both DEV & testing activities

- **Localization** ($L_{10}n$)
  - Also called $L_{10}n$, the subscript 10 is used to indicate that there are 10 characters between "L" and "n" in the word Localization.
  - Translation of all product messages and documentation
  - Done by language experts
  - Includes both DEV & testing activities
- **Globalization**
  - Not very popular
  - Also called $\mathbf{G_{12}n}$
  - Internationalization includes localization but some companies want to separate as the team that does both are different, and hence this term

GLOBALIZATION= INTERNATIONALIZATION+ LOCALIZATION

Test Phases for Internationalization Testing :- Testing for Internationalization requires a clear understanding of all activities involved and their sequence. The job of testing is to ensure the correctness of activities done earlier by other teams.

The Major Activities in Internationalization Testing

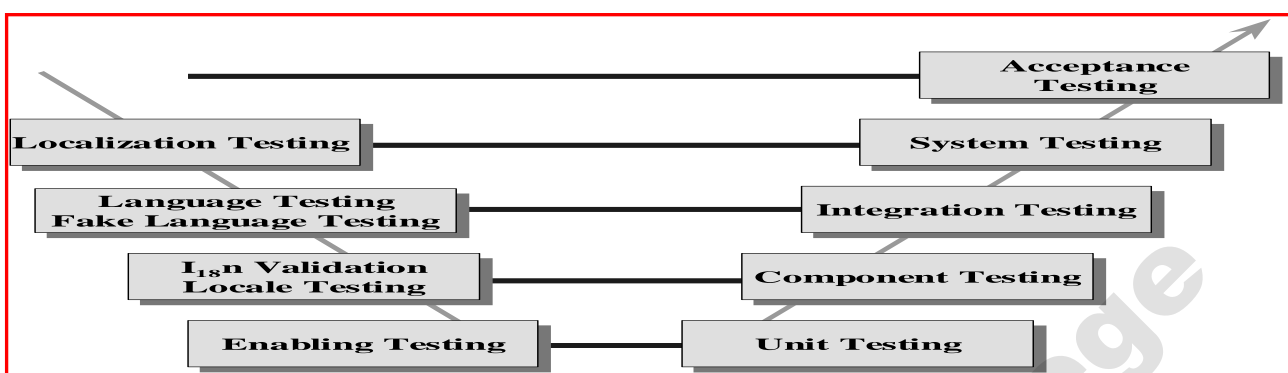


The Testing for internationalization is done in multiple phases in the project life cycle. The diagram below Elaborates the SDLC V model described and how the different phases of this model are related to various I18n testing Activities. Enabling testing is done by the developer as a part of the Unit testing Phase.

Some Important Aspects of Internationalization testing are:-

Arunai Engineering College

1. Testing the code for how it handles input, strings and sorting items;
2. Display of Messages for Various Languages; and
3. Processing of Messages for Various Languages and Conventions.



Enabling Testing:-

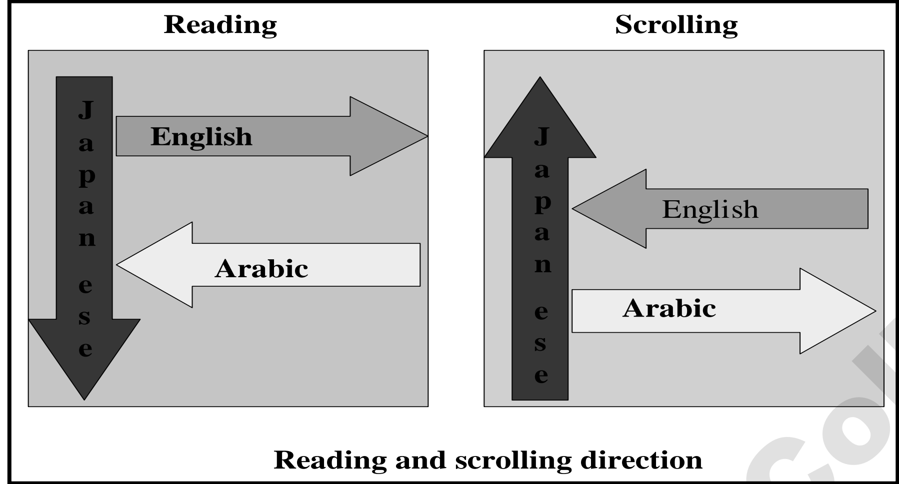
Enabling Testing is a white box testing methodology , which is done to ensure that the source code used in the software allows internationalization. A source code , which has hard coded currency format and date format, fixed length GUI screens or dialog boxes, read and print messages directly on the media is not considered enabled code. ***An activity of code review or code inspection mixed with test cases for unit testing, with an objective to catch*** ***$I_{18}n$ defects is called enabling testing.*** The year 2000 is a classic I18n defect. Enabling testing finds the majority of $I_{18}n$ defects If this is not done in the unit test phase, exponential effort has to be spent in later phases as it impacts code, design, etc. Also other $I_{18}n$ testing for fake language, $l_{10}n$ has to be repeated.

Enabling Testing – Checklist:-

- Find out those APIs/function calls that can't be used for I18n (printf, scanf) – NLSAPI, unicode, GNU gives some APIs instead
- Check the code for hard-coded date, currency format, ASCII usage or character constants.
- Check the code for arithmetic operations on date ie there is no computations (additions and subtractions) done on date variables or different format forced to the date in the code.
- Check that no format is forced to date field
- Check each field in the screen for extra space (normally 50% extra space is allotted)
- Ensure that region-based messages/slang are not used (e.g., Hi, references to colour)
- Ensure no string operations are performed on the code (substring search, concatenation); only APIs provided by I18n are to be used
- The code does not assume any predefined path, filename, directory name in NLS directory
- Check code doesn't make any assumptions about bit representation (8, 16, 32), and bit operations are not used
- Ensure adequate length is allocated to accommodate translated messages
- Check that pictures, logos and bitmaps do not have embedded text
- Ensure that all messages have code in-line comments for helping translators (e.g. pre-ponement of meeting)
- Ensure all resources are (dialog boxes, screen shots, bitmaps, etc.)

Arunai Engineering College

- Ensure technical jargons are not used and that the text may be understood by even the least-skilled user (e.g., pipe overflow)
- Ensure that the directions of reading / writing are opposite to scrolling, and that they follow the language convention



Reading and scrolling direction

Locale Testing:-

Locale Testing is not as elaborate procedure as enabling testing. The focus of Locale testing is Limited to :-

- Changing the different locale using the system settings or environment variables, and testing the software functionality, number, date, time and currency format is called locale testing.
- It is to used validate the effects of locale change in the product. A locale change affects date, currency format, the display of items in the screen, dialog boxes and the text.
- Black box methodology tests all component features for each locale.
- In Microsoft Windows 2000, you can change locale by clicking "Start->Settings->Control Panel->Regional options (demo).

Locale Testing focuses on testing the conventions for number, punctuations, date and time, and currency format.

Locale Testing - Checklist

- All features that are applicable to I18n are tested with different locales of the software for which it is intended.
- Some of the activities that need not be considered for I18n testing are auditing, debug code, log of activities and such features that are used only by English administrators and programmers.
- Hot keys, function keys and help screens are tested with different applicable locales (this is to check whether locale change would affect the keyboard settings).
- Date and time format is in line with the defined locale of the language. For example if the US English locale is selected, the software should display data in mm/dd/yyyy date format.
- Currency is in line with the selected locale and language. For example, currency should be AUS$ if the language is AUS English.
- Number format is in line with the selected locale and language. For example, the correct decimal punctuations are used and the punctuation is put at the right places.

Arunai Engineering College

- Time zone information and daylight saving time calculations (if used by the software) are consistent and correct.

Internationalization Validation:-

Objectives of $I_{18}n$ the validations is performed with the following objectives:-

1. The software is tested for functionality with ASCII, DBCS, and European characters•
2. The software handles string operations, sorting, sequencing operations as per the language and characters selected
3. The software display is consistent with characters that are non-ASCII in GUI and menus
4. The software messages are handled properly

$I_{18}n$ Validation – Input Method Editor

This is a soft keyboard used to enter non-English characters into the product. IME soft keyboard for Japanese.

$I_{18}n$ validation – Checklist:-

1. The functionality in all languages and locales are the same.
2. Sorting and sequencing the items are as per the conventions of language and locale. For example if $ is mentioned as the currency symbol for USA, sorting should take care of symbol & punctuations.
3. The input to the software can be in non-ASCII (Use of tools such as IME) and functionality is consistent with non-ASCII.•
4. The non-ASCII characters in the name are displayed as they were entered.
5. The cut or copy -and-paste of non-ASCII characters retains their style after pasting, and the software functions as expected.
6. The software functions correctly with different languages / words / names generated with IME and other tools; for example, Login should work with an English user name as well as with a German user name with some accented characters.
7. The documentation contains consistent documentation style and punctuations, and all language / locale conventions are followed for every target audience.

Arunai Engineering College

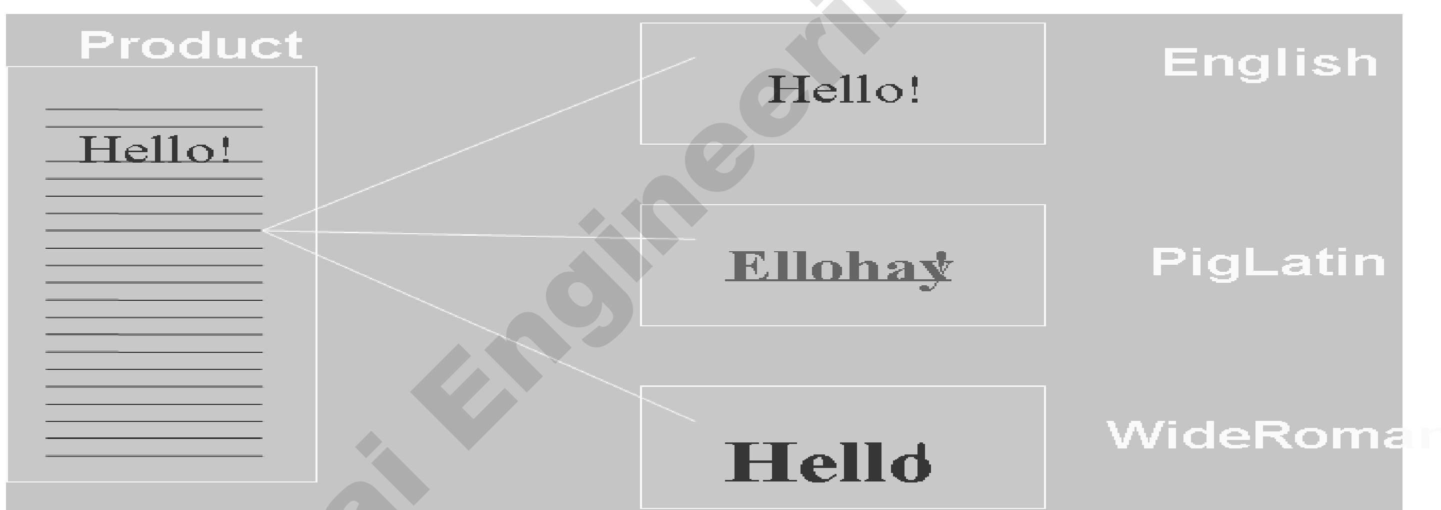
8. All the runtime messages in the software are as per the language, country terminology and usage along with proper punctuations; for example, the currency amount 123456789.00 should get formatted as 123,456,789.00 in the US and as 12,34,56,789.00 in India)

I18 *n* Validation Focuses on component functionality for Input/ Output of Non English Messages.

Fake Language Testing

Fake Language testing uses software translators to catch the translation and localization issues early. This also ensures that switching between languages works properly and correct messages are picked up from proper directories that have the translated messages. Fake Language testing helps in identifying the issues proactively before the product is localized. For this purpose , all messages are consolidated from the software , and fake language conversion are consolidated from the software, and fake language conversion are done by tools and tested. The Fake language translators use English like Target Languages, which are easy to understand and test. This type of testing helps English testers to find the defects that may otherwise found only by Language Experts during Localization Testing .

In the figure there are two English like Fake Languages used (Pig Latin and Wide Roman) A message in the program, "Hello" as "Ellohay" in Pig Latin and "Hello" in Wide Roman .This helps in identifying whether the proper target language has been picked up by the software when language is changed dynamically using system setting .



The Following items in the checklist can be used for Fake Language Testing:-

1. Ensure the software functionality is tested for at least one of the European single byte fake languages (e.g., Pig Latin) Ensure the software functionality is tested for at least one double byte language (e.g., Wide Roman)
2. Ensure all strings are displayed properly in the screen
3. Ensure the screen width, size of pop-ups and dialog boxes are adequate for string display with the fake languages.

Fake Language testing helps in simulating the functionality of the localized product for a different language using software translator.

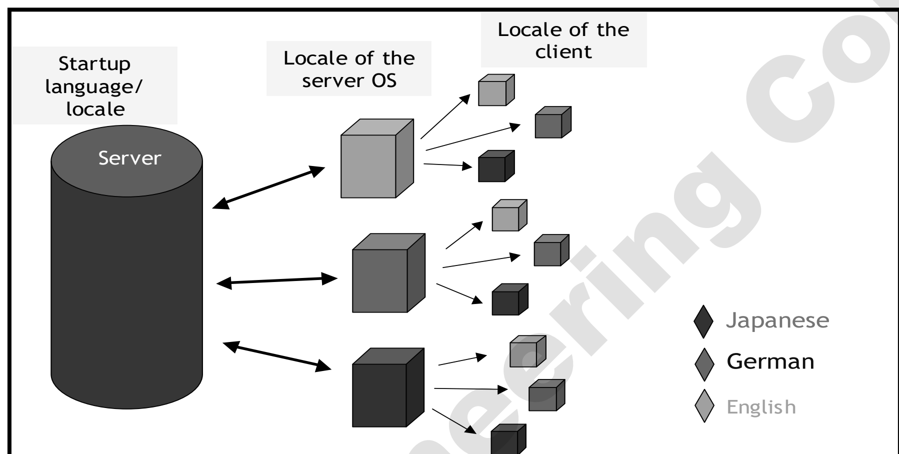
Arunai Engineering College

Language Testing:-

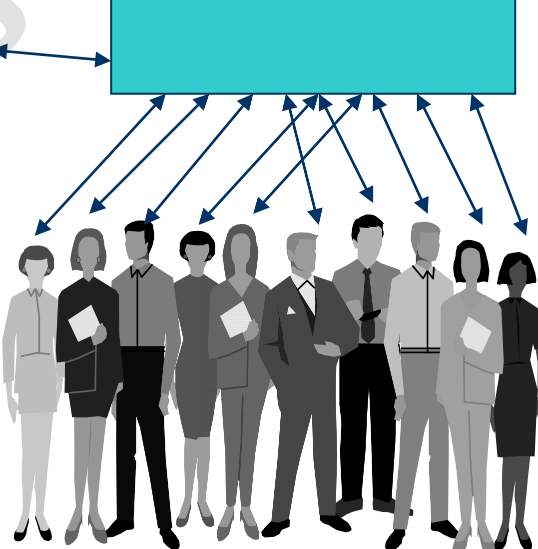
- Short form of "language compatibility testing"
- This testing is done to ensure that on other language settings the functionality of the software is not broken and that it is still compatible across the network.
- When data is transmitted between machines or between softwares and operating systems, the code page, bit stream, message conversions taking place for internationalization.

Language Testing - Checklist

- *Check the functionality on one English, one non-English and one double-byte language platform combination.*
- *Check the performance of key functionality on different language platforms and across different machines connected in the network.*



I speak only English but can deal with anyone

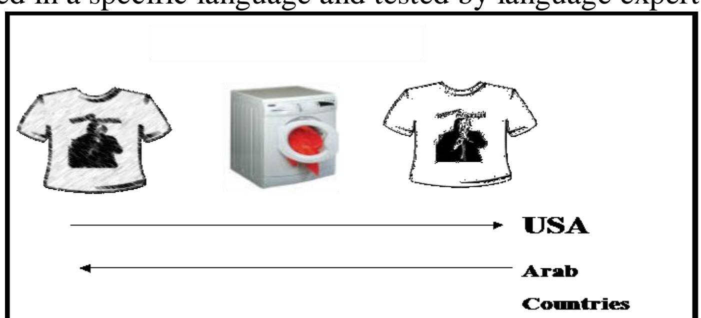


Localization Testing :-

1. Build tools consolidate all messages.
2. Documents and other artifacts are collected.
3. They are sent to language experts for translation.

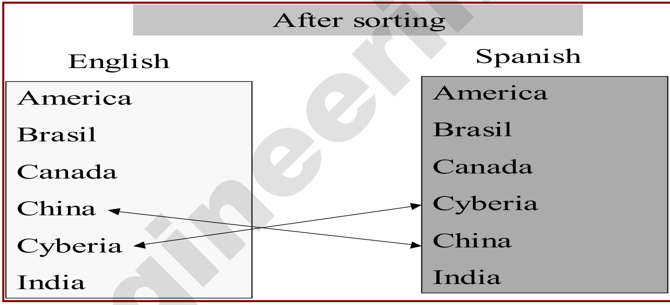
Arunai Engineering College

4. Process of localization is expensive.
5. Not all messages, documents need to be localized.
6. Process of localization also alters the GUI screens, dialog boxes, icons and bitmaps.
7. Process of customization.
8. The product is installed in a specific language and tested by language experts.



<u>Localization Testing – Checklist</u>

1. *All the messages, documents, pictures, screens are localized to reflect the native users and the conventions of the country, locale and language.*
2. *Font sizes and hot keys are working correctly in the translated messages, documents and screens.*
3. *Filtering and searching capabilities of software work as per the language and locale conventions.*
4. *Addresses, phone numbers, numbers and postal codes in the localized software are as per the conventions of the target user.*
5. *Sorting and case conversions are right as per language convention; for example, sort order in English is A, B, C, D, E, whereas in Spanish the sort order is A, B, C, CH, D, E.*



Sort Order in English and Spanish

<u>Tools Used For Internationalization :-</u>

There are several tools available for internationalization. These largely depend on the technology and platform used. For Example, the tools used for client server technology is different from those for web services technology using Java.

S.No	Sample list of tools for Microsoft OS	Name of tool for Linux OS	Purpose
1	MS localization Studio	GNU gettxt ()	Enabling and enabling testing
2	http://BabelFish.Altavista.com	http://BabelFish.Altavista.com	Fake language testing
3	MS regional settings	LANG and set of env variables	Locale testing
4	IME	Unicode IME (several variants exist from several companies)	I18n validation
5	http://www.snowcrest.net/donnelly/piglatin.html	http://www.snowcrest.net/donnelly/piglatin.html	Fake language testing Pig Latin)

Arunai Engineering College

ADHOC TESTING :-

- **Overview**
- ***Ad hoc* testing Vs planned testing**
- **Buddy testing**
- **Pair testing**
- **Exploratory testing**
- **Iterative testing**
- **Agile & extreme testing**
- **Defect seeding**
- **Defect bash**
- **Drawbacks of *ad hoc* testing**

All types of testing explained earlier are part of planned testing and are carried out using certain specific techniques (Boundary value Analysis) there are family of test types which are carried out in un planned manner hence it is named Adhoc Testing. Related Type of Adhoc Testing are

1. Buddy Testing
2. Exploratory Testing
3. Pair Testing
4. Iterative Testing
5. Agile and Extreme Testing
6. Defect Seeding

Issues of planned testing

a. Goes by level of understanding at the time of design
b. Validation of test cases happens at runtime
c. Lack of clarity on requirements impacts quality
d. Lack of skills affects quality of test cases
e. Lack of time for design affects completeness

- After some of the Planned test cases are executed, the clarity on the requirement improves. Test cases written earlier may not reflect the better clarity gained in this process.
- After going through a round of planned test execution, the skills of the test engineers becomes but the test cases may not have been updated to reflect the improvement in skills.
- The lack of time for test design affects the quality of testing , as there could be missing perspectives.
- Planned Testing Enables catching certain types of defects. Though Planned tests help in boosting the testers Confidence , it is the testers "intuition" that often finds critical defects.

Definition: *Ad Hoc* Testing

Testing done without using any formal testing technique is called *ad hoc* testing.

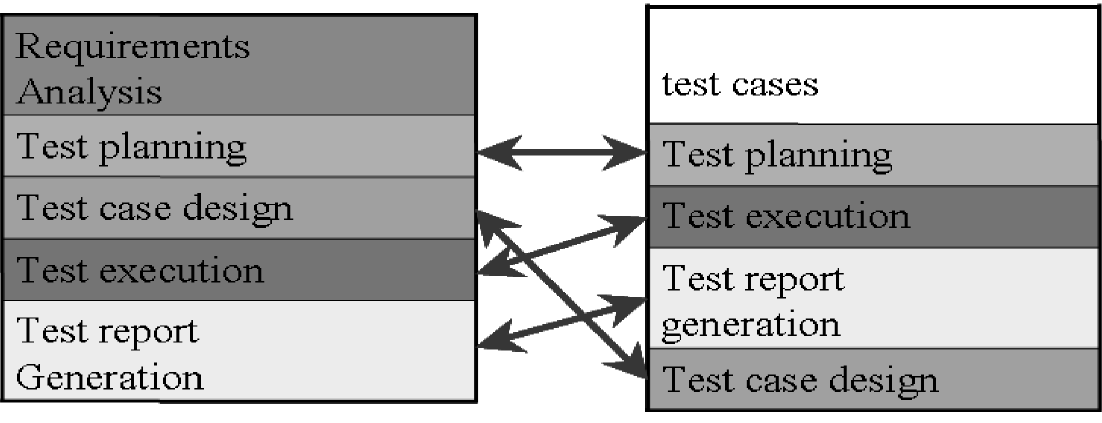
Pesticide Paradox

One of the Principles of Software Testing explains the situation where the surviving pests in a farm creates resistance to a particular pesticide. The situation requires the farmer to use a different types of pesticide every time for the next crop cycle. Similarly, products defects gets tuned to planned test cases and those test cases may not uncover defects in the nest test cycles unless new perspectives are added. Planned Test Cases requires constant updates, sometimes even on a daily basis, incorporating the new learning. Updating test cases very frequently may become time consuming and tedious job .In such cases we have to follow Adhoc Testing.

Adhoc Testing Versus Planned Testing :-

Testing done without using any recognized testing technique is called ad hoc testing.

Arunai Engineering College



Constant interaction with developers and other project team members may lead to better understanding of the product from various perspectives. Since Adhoc tests require better understanding of the product, it is importance to stay "Connected".

Due to lack of communication, change in the requirements may not be informed to the test team. When test Engineer does not know the requirements changes, it is possible to miss few tests. This may result in a few undetected defects. It is possible to unintentionally miss some perspectives due to changed requirements.

Interaction with developers and other team members may help in getting only a set of perspectives. These type of interaction may bias the testing team. Hence it is important to constantly question the test cases and also interact with people outside the organization to find different ways of using the product and use them in adhoc testing.

Adhoc testing can be performed on a product at any time, but the return from adhoc testing are more if they are run after running planned test cases. Adhoc testing can be planned in one of two ways:-

1. After a Certain number of planned test cases are executed. In this case, the product is likely to be in a better shape and thus newer perspectives and defects can be uncovered. Since Adhoc testing does not require all the test cases to be documented immediately, this provides an opportunity to catch multiple missing perspectives with minimal time delay.
2. Prior to planned testing. This will enable gaining better clarity on requirement and assessing the quality of the product upfront.

Drawbacks of Adhoc Testing and Their Resolutions:-

Drawback	***Possible resolution***
Difficult to ensure the perspectives covered in ad hoc testing are used in future	• Document ad hoc tests after test completion
Large number of defects found in ad hoc testing	• Schedule a meeting to discuss defect impacts • Improve the test cases for planned testing
Lack of comfort on coverage of ad hoc testing	• When producing test reports combine the planned test and ad hoc test • Plan for additional planned test and ad hoc test cycles

Arunai Engineering College

Difficult to track the exact steps done	• Write detailed defect reports in a step-by-step manner • Document ad hoc tests after test execution
Lack of data for metrics analysis	• Plan the metrics collection for both planned tests and ad hoc tests, not only for ad hoc testing

Buddy Testing :-

Def: A developer and tester working as buddies to help each other on testing and in understanding the specifications is called Buddy Testing

- Two team members (developer and a tester)are identified as buddies. The buddies mutually help each other, with a common goal of identifying defects early and correcting them
- This good working relationship as buddies overcome fear.
- Budding people with good working relationships yet having diverse backgrounds is a kind of a safety measure that improves the chance of detecting errors in the program very early
- Buddies should not feel mutually threatened or get a feeling of insecurity during buddy testing. They are trained on the philosophy and objective of buddy training.
- They also have to agree on the modalities and the terms of working before actually starting the testing work. They stay close together to be able to follow the agreed plan
- The Buddy can check for compliance to coding standards , appropriate variable definitions , missing code, sufficient inline code documentation , error checking.
- Buddy testing uses→ both white box and black box testing approaches.
- after testing →generates specific review developers.
- The more specific the feedback, easier it is for the developer to fix the defects . The buddy may also suggest ideas to fix the code when pointing out an error in the work product. A buddy test may help avoid errors of omission, misunderstanding, and miscommunication by providing varied perspectives or interactive exchanges between the buddies,
- Buddy testing not only helps in finding errors in the code but also helps the tester to understand how the code is written and provides clarity on specifications. Buddy testing is normally done at the unit phase , where there are both coding and testing activities .

Pair Testing:-

Pair testing is testing done by two testers working simultaneously on the same machine to find defects in the product

.**Example:-**

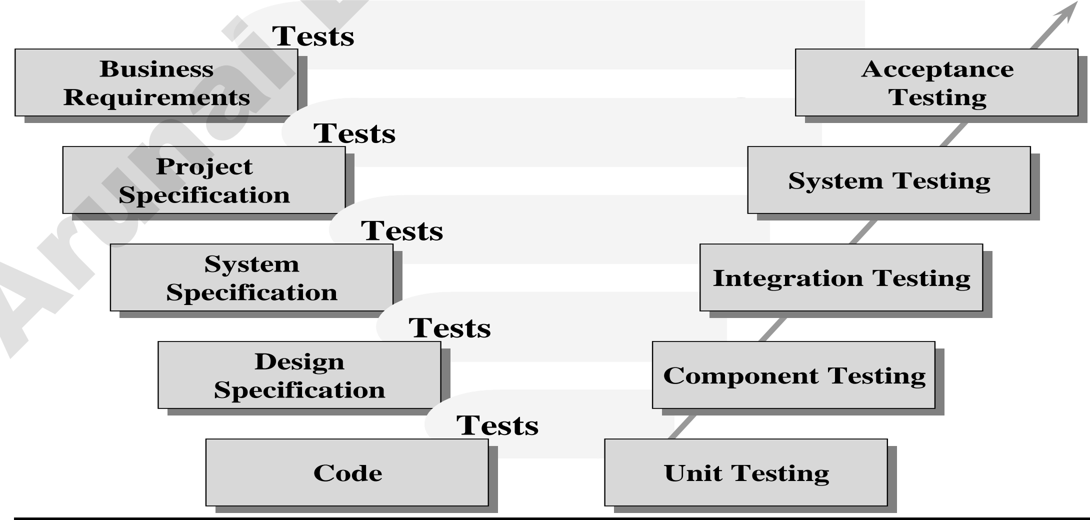
Arunai Engineering College

For e.g., two people traveling in a car to find a new place

- Two testers pair up to uncover new defects
- One person executes tests, and the other person observes
- Rotation of roles
- A session of one or two hours
- A senior person and a junior person make an ideal pair

- Pair testing takes advantage of the concept of the presence of one senior member can also help in pairing; this can cut down on the time spent on the learning curve of the product. It enables better training to be given to the team members; The impact of the requirements can be fully understood and explained to less experienced individuals.
- Pair testing can be done during any phase of testing. It encourages idea generating right from the requirements analysis phase, taking it forward to the design, coding and testing phases .
- Testers can pair together during the coding phase to generate various ideas to test the code and various components.
- After completion of component testing, during integration, tester can be paired to test the interfaces together. Pair testing during system testing ensures that product level defects are found and addressed.
- When the product is in new domain and not many people have the desired knowledge pair testing will be useful. Pair testing can track that vague defect that is not caught by a single person testing,
- A defect found during such pair testing may be explained better by representation of two members. Pair testing is extension of the "Pair Programming" concept used as a technique in the extreme programming model.
- Pair testing require interaction and exchange of ideas between two individuals. Team members pair with different persons during project life cycle, the entire project team can have a good understanding of each other ,

Situation when Pair Testing Becomes Ineffective:-



During pairing, teaming up individual high performers may lead to problem may be possible that during the course of the session , one person takes the lead and other has a laid back attitude . This may not produce the desired

Arunai Engineering College

results. In case the pair of individuals in the team are ones who do not try to understand and respect each other, pair testing may lead to frustration and domination. When one member is working on the computer and other is playing the role of scribe , if their speed of understanding and execution does not match, it may result in loss of attention. It may be difficult in the later stage.

Pairing up juniors with experienced members may result in the members may result in the former doing tasks that the senior may not want to do, At the end of the session, there is no accountability on who is responsible for steering the work, providing directions and delivering the results.

Exploratory Testing :-

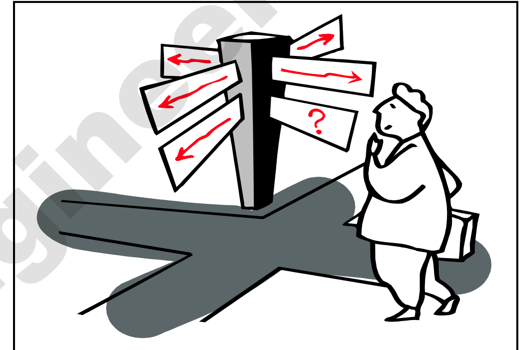
Technique used to find defects in Adhoc testing is to keep exploring the products, covering more depth and breadth. Exploratory testing tries to do that with specific objectives, tasks and plans. Exploratory testing can be done during any phase of testing.

Exploratory testers may execute their test based on their past experiences in testing a similar product, or a product of similar domain, or a product in a technology area. Exploratory testing can be used to test software that is untested, unknown , or unstable. It is used when it is not obvious what the nest test should be and or when we want to go beyond the obvious tests.

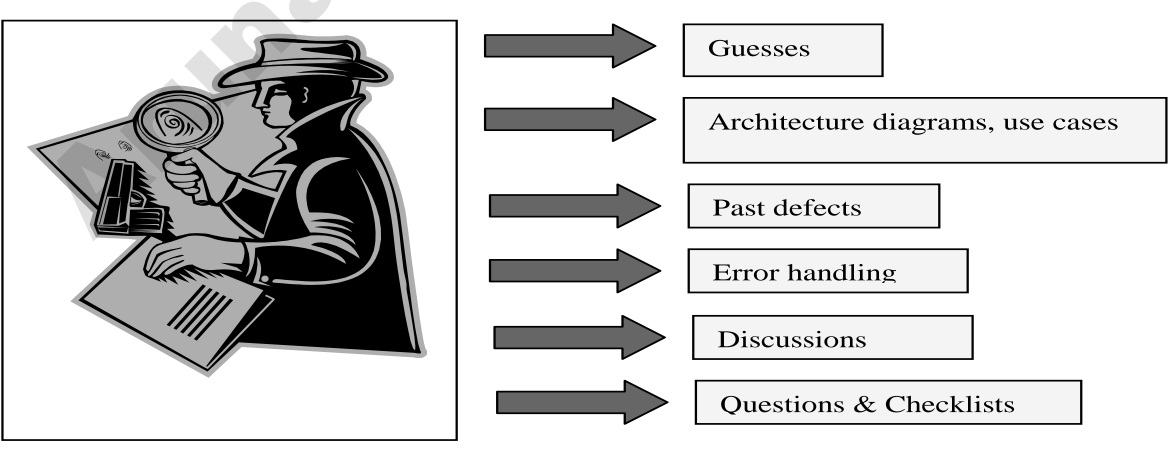
Exploratory Testing Techniques:-

For e.g., driving the car in a new area. Common techniques used to reach the destination is

- Getting a map
- Asking pedestrians
- Random direction and search
- Calling up a friend
- Enquiring at gas stations
- Looking at boards / signs



Exploratory Test Techniques



Engineering College

Guesses are used to find the part of the program that is likely to have more errors. Because a tester would have already faced situations to test a similar product ort software. Those tests from guesses are used on the product to check for similar defects,

Architectural Diagrams and Use Cases depicts the interactions and relationships between different components and modules . us eases give an insight of the product's usage from the end users perspectives. Use case can explain a set of business events , the input requires, people involved in those events and the expected output.

Study of Past Defects studying defects reported in the previous releases helps in understanding of the error prone functionality / modules in a product development environment .

Error Handling is the product in another technique to explore. Error handling is a portion of the code which prints appropriate messages or provides appropriate action in case of failures. We can check using exploratory test for various scenarios for graceful error handling. For Example in the case of a catastrophic error, termination should be with a meaningful error message. Error Handling provides a message or corrective action in such situations. Test can be performed to simulate such situations to ensure that the products code take care of these aspects.

Discussion – Exploration may be planned based on the understanding of the system during project discussions or meetings. Plenty of information can be picked up during these meetings regarding implementation of different requirements for the products. They can be noted and used while testing.

Questionnaires and Checklists to perform the exploration. Questions like "What, When, How, Who and Why" can provide leads to explore areas in the product. To understand the implementation of functionality in a product , open-ended questions like "What does this module do", " When is it being called or used?", " how is the input processed" ," who are the users of this modules", etc.

<u>Iterative Testing :-</u>

For e.g., a person driving without a map trying to count the restaurants in a town

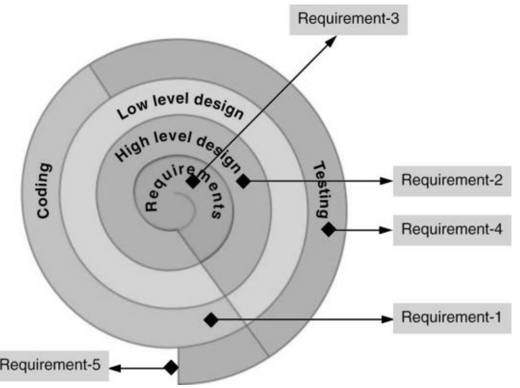
Customer will have a usable product at the end of every iteration .It is possible to stop the product development at any particular iteration and market the product as an independent entity.

Customer and Management can notice the impact of defects and the product functionality at the end of each iteration. They can take a call to proceed to the next level or not, base don the observations made in the last iterations. A test plan is created at the beginning of the first iterations and update for every subsequent iterations. This can be broadly defined the type and scope of testing to be done for each of the iterations. Developers create unit test cases to ensure that the

Arunai Engineering College

program developed goes through complete testing. Unit test cases are also generated from black box perspective to more completely test the product. Regression Testing may be repeated at least every alternative iterations so that the current functionality is preserved since iterative testing involves repetitive test execution of tests that were run fro the previous iterations, it becomes a tire some exercise for the testers.

Assume that a defect was found in the second iteration and was not fixed until the fifth. There is a possibility that the defect may no longer be valid or could have become void due to revised requirements during the third, fourth and fifth iterations. In the example above the counting the number of restaurants starts from the first road visited , the results of the search can be published at the end of each iteration and released.



- Each of the requirements is at a different phase
- Testing needs to focus on the current requirement
- It should ensure that all requirements continue to work
- More re-testing effort

Agile and Extreme Testing :-

Call Attendant: Our process requires the person for whom the certificate is issued to come and sign the form.

Caller: I understand your process, but I am asking for the death certificate of my grand father.

Agile and Extreme (XP) models take the processes to the extreme to ensure that customer requirements are met in a timely manner. Customer partner with the project teams to go step by step in bringing the project to completion in a phased manner. The customer becomes part of the project team so as to clarify any doubts/questions.

Agile and Extreme (XP) methodology emphasizes the involvement of the entire team, and their interactions with each other, to produce workable software that can satisfy a given set of features. As a result of such interactions, all ideas are exchanged. Software is delivered as a small release with features being introduced in increments.

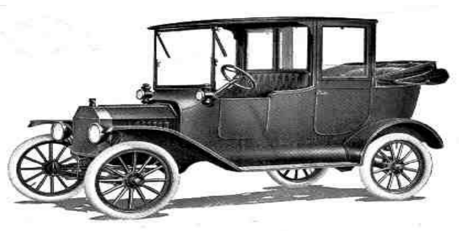
A typical XP project day start with a meeting called the Stand Up meeting . At the start of each day, the team meets to decide on the plans of actions for the day. During this meeting the team brings up any clarifications or concerns. They are discussed and resolved. The entire team gets a consistent view of what each team members is working on. Tester present to the project team the progress of the project based on the test results.

Arunai Engineering College

Policies / concepts of Agile and Extreme

- Cross boundaries
- Incremental progress – both product and process evolve in incremental way
- Travel light – least overhead
- Communicate – more focus on communication
- Write tests before coding – all unit tests run at 100%
- Make frequent small releases
- Involve customers all the time

Example for extreme testing

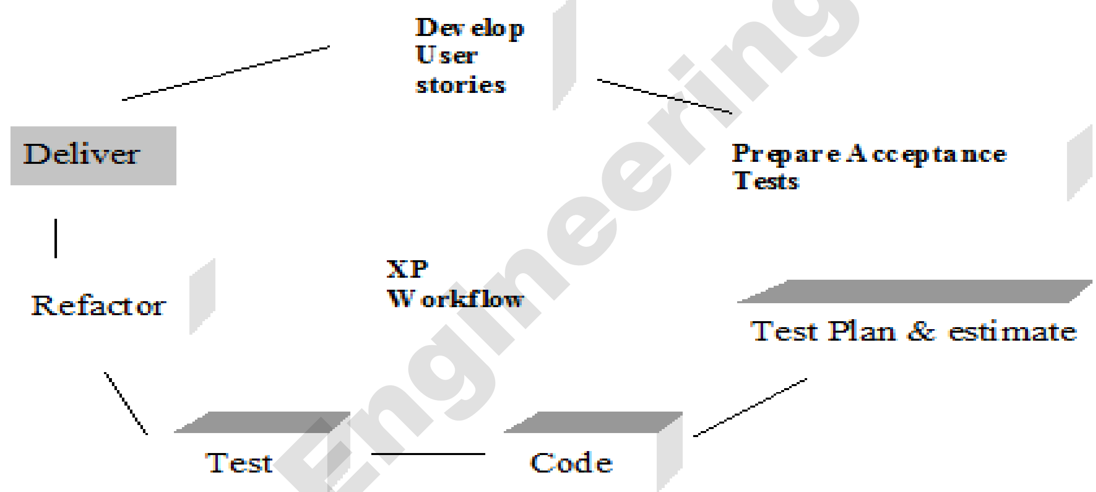


For e.g.,

- Basic features : wheel, brake, pedal, tyres
- New features added incrementally
- Every year / every manufacturer release new models many times in a year with new features
- Thus Automobile industry keeps growing & Improving

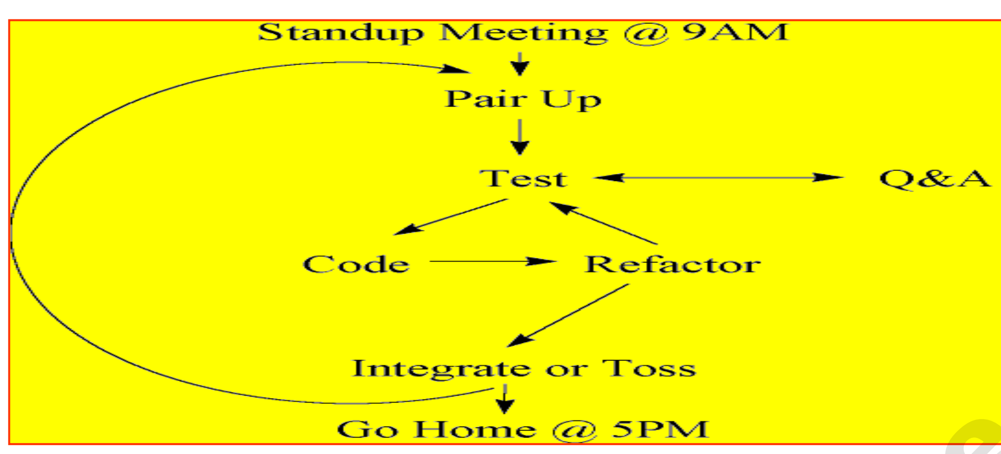
- **Technically, driving a car using a joystick is easier , But customers are comfortable with steering wheels**

The basic steps that are carried out :-

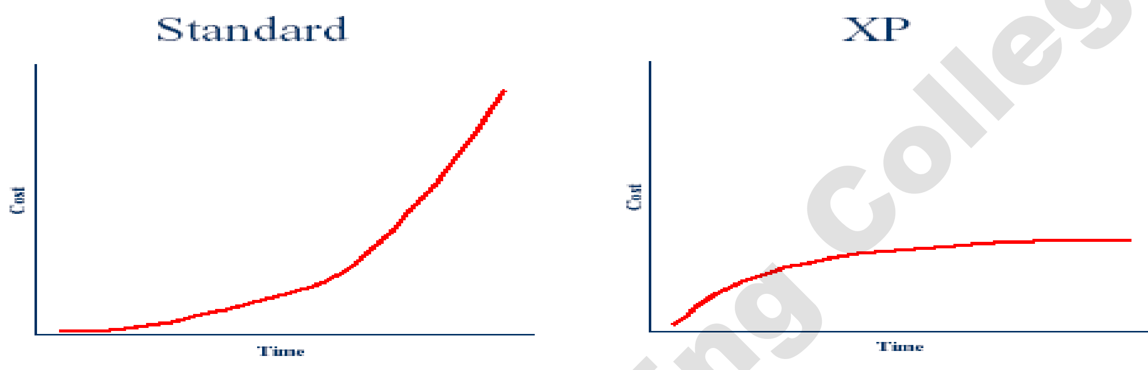


- Develop and Understand User Story
- Prepare acceptance tests
- Test Plan and Estimation
- Code
- Test
- Refactor
- Automate
- Accepted and Delivered

Arunai Engineering College



Cost of Change:-



Defect Seeding:-

Def: Defect seeding is a method of intentionally introducing defects into a product to check the rate of detection and residual defects.

Error Seeding is also known as *Debugging* . It acts as a reliability measure for the release of the product. Usually one group members in the project injects the defects while an other group tests to remove them. The purpose of this exercise is while finding the known seeded defects, the unseeded/ un earthed defects may also be uncovered . Defects that are seeded are similar to real defects. Defects that can be seeded may vary from sever or critical defects to cosmetic errors. Defect Seeding may act as a guide to check the efficiency of the inspection or testing process. It serves as a confidence measure to know the percentage of defects removal rates. It acts as a measure to estimate the number of defect yet to be discovered in the system.

Defects that can be seeded may vary from severe or critical defects to cosmetic errors.

- For example : a team seeds 20 defects, and testing finds out 12 seeded defects and 25 other defects

Total latent defects = (defects seeded / defects seeded found) * Other defects found

- 20 / 12 * 25 = 41.67 = 42

Based on the above calculation , the number of estimated defects yet to be found is 42.

Arunai Engineering College

When a group knows that there are seeded defects in the system it acts as a challenge for them to find as many of them as possible. It adds a new energy into their testing .in case of manual testing, defects are seeded before the start of the testing process. When the tests are automated, defects can be seeded any time .

It may be useful to look at the following issues on defect seeding as well.

1. Care should be taken during the defect seeding process to ensure that all the seeded defects are removed before the release of the product.
2. The code should be written in such a way that the errors introduced can be identified easily, Minimum number of lines should be added to seed defects so that the effort involved in removal becomes reduced.

- It is necessary to estimate the effort required to clean up the seeded defect along with effort for identification. Effort may also be needed to fix the real defects

ALPHA, BETA TESTS

- Goal : allow users to evaluate the software in terms of clients expectations and goals.
- The acceptance tests must be planned carefully with input from the client/users. Acceptance test cases are based on requirements.
- The user manual is an additional source for test cases. System test cases may be reused.
- The software must run under real-world conditions on operational hardware and software.
- For continuous systems the software should be run at least through a 25-hour test cycle.
- Development organizations will often receive their final payment when acceptance tests have been passed.
- Acceptance tests must be rehearsed by the developers/testers. There should be no signs of unprofessional behavior or lack of preparation. Clients do not appreciate surprises. They should be provided with documents and other material to help them participate in the acceptance testing process, and to evaluate the results
- After acceptance testing the client will point out to the developers which requirement have/have not been satisfied. Some requirements may be deleted, modified, or added due to changing needs.
- If the client is satisfied that the software is usable and reliable, and they give their approval, then the next step is to install the system at the client's site. If the client's site conditions are different from that of the developers, the developers must set up the system so that it can interface with client software and hardware. Retesting may have to be done to insure that the software works as required in the client's environment. This is called installation test.
- If the software has been developed for the mass market , then testing it for individual clients/users is not practical or even possible in most cases. Very often this type of software undergoes
- two stages of acceptance test.
    - alpha test. → test takes place at the developer's site. A cross-section of potential users and members of the developer's organization are invited to use the software. Developers observe the users and note problems.

Arunai Engineering College

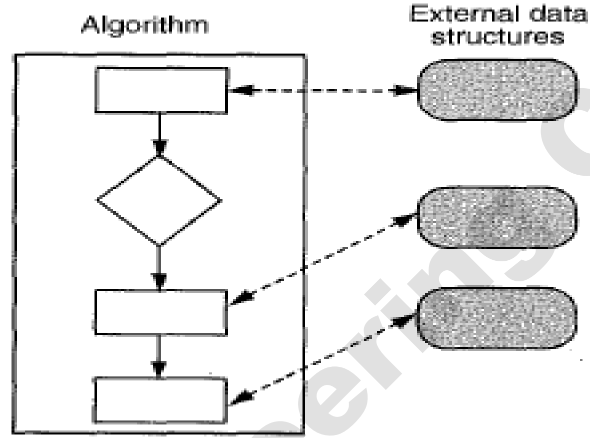
- Beta test→ sends the software to a cross-section of users who install it and use it under real world working conditions. The users send records of problems with the software to the development organization where the defects are repaired sometimes in time for the current release. In many cases the repairs are delayed until the next release.

TESTING OO SYSTEMS :-

In procedure-oriented languages

Algorithms +Data Structures =Programs.

These programming languages were algorithm-centric in that they viewed the program as being driven by an algorithm that traced its execution from start to finish, as shown in Figure Data was an external entity that was operated upon by the algorithm.



Conventional algorithm centric programming languages.

Fundamentally, this type of programming languages was characterized by

1.Data being considered as separate from the operations or program and

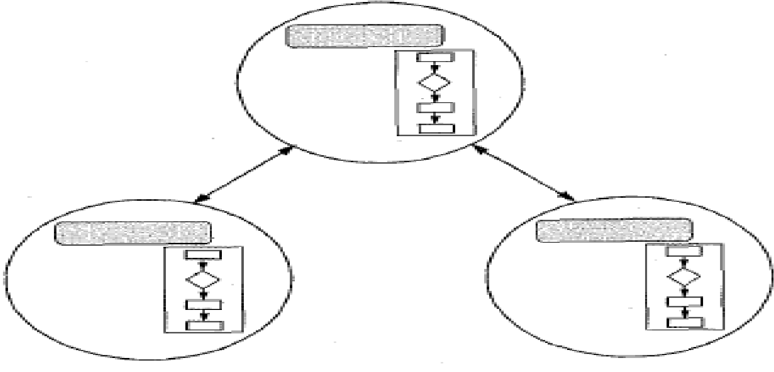
2.Algorithm being the driver, with data being subsidiary to the algorithm.

In OO languages

There are two fundamental paradigm shifts in OOlanguages and programming:

First→ the language is data- or object-centric.

Second→ The data and the methods that operate on the data go together as one indivisible unit.



Arunai Engineering College

Object centric language-algorithm and data tightly coupled.

Some of the basic concepts of OO systems are relevant for testing

Classes :Classes form the fundamental building blocks for OO systems. A class is a representation of a real-life object. Each class (or the real-life object it represents) is made up of attributes or variables and methods that operate on the variables.

```
Class rectangle
{
private int length, breadth;
public:
new (float length, .float. breadth)
(
this->length = length;
this->breadth = breadth;
float area ()
{
return (length*breadth1;
return (2*(length+breadth))
}
} ;
```

Objects

Objects are the dynamic instantiation of a class. Multiple objects are instantiated using a given (static) class definition. Such specific instantiations are done using a constructor function.

Constructor

A constructor function brings to life an instance of the class. Each class can have more than one constructor function. Depending on the parameters passed or the signature of the function, the right constructor is called.

Encapsulation

Encapsulation provides the right level of abstraction about the variables and methods to the outside world.

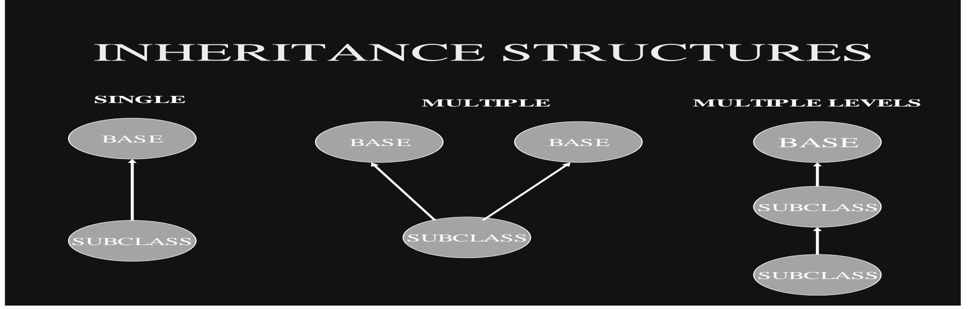
Polymorphism

This property of two methods-in different classes-having the same name but performing different functions is called polymorphism.

Inheritance

Inheritance enables the derivation of one class from another without losing sight of the common features. This ability is called inheritance. The original class is called the parent class (or super-class) and the new class is called a child class (or derived class, or sub-class). Inheritance allows objects (or at least parts of the object) to be reused. A derived class inherits the properties of the parent class-in fact, of all the parent classes, as there can be a hierarchy of classes. Thus, for those properties of the parent class that are inherited and used as is, the development and
testing costs can be saved.

Arunai Engineering College



DIFFERENCES IN OO TESTING

From a testing perspective, the implication is that testing an oo system should tightly integrate data and algorithms .. The dichotomy between data and algorithm that drove the types of testing m procedure-oriented languages has to be broken. Testing OO systems broadly covers the following topics.

1. Unit testing a class
2. Putting classes to work together (integration testing of classes)
3. System testing
4. Regression testing
5. Tools for testing OO systems

Unit Testing a Set of Classes

Classes are the building blocks for an entire OO system.

Why classes have to be tested individually first:

reasons:

1. A class is intended for heavy reuse. A residual defect in a class can therefore, potentially affect every instance of reuse.
2. Many defects get introduced at the time a class (that is, its attributes and methods) gets defined. A delay in catching these defect makes them go into the clients of these classes. Thus, the fix for the defect would have to be reflected in multiple places, giving rise to inconsistencies.
3. A class may have different features; different clients of the class may pick up different pieces of the class. No one single client may use all the pieces of the class. Thus, unless the class is tested as a unit first, there may be pieces of a class that may never get tested.
4. A class is a combination of data and methods. If the data and methods do not work in sync at a unit test level, it may cause defects that are potentially very difficult to narrow down later on.
5. Unlike procedural language building blocks, an OO system has special features like inheritance, which puts more "context" into the building blocks. Thus, unless the building blocks are thoroughly tested stand-alone, defects arising out of these contexts may surface, magnified many times, later in the cycle.

Conventional methods that apply to testing classes

Some of the methods for unit testing that we have discussed earlier apply directly to testing classes. For example:

1. Every class has certain variables. The techniques of boundary value analysis and equivalence partitioning discussed in black box testing can be applied to make sure the most effective test data is used to find as many defects as possible.
2. As mentioned earlier, not all methods are exercised by all the clients, The methods of function coverage that were discussed in white box testing can be used to ensure that every method (function) is exercised.
3. Every class will have methods that have procedural logic. The techniques of condition coverage, branch coverage, code complexity, and so on that we discussed in white box testing can be used to make sure as many branches and conditions are covered as possible and to increase the maintainability of the code.
4. Since a class is meant to be instantiated multiple times by different clients, the various techniques of stress testing.

Arunai Engineering College

Integration testing

Since OOsystems are ·designed to be made up of a number of smaller components or classes that are meant to be reused (with necessary redefinitions), testing that classes work together becomes the next step, once the basic classes themselves are found to be tested thoroughly. In the case of OO systems, because of the emphasis on reuse and classes, testing this integration unit becomes crucial. in an OO system, the way in which the various classes communicate with each other is through messages. A message of the format

<instance name>.<method name>.<variables>

calls the method of the specified name, in the named instance, or object (of the appropriate class) with the appropriate variables.
Methods with the same name perform different functions → polymorphism. From a testing perspective, polymorphism is especially challenging because it defies the conventional definition of code coverage and static inspection of code.

The various methods of integration

- top-down
- bottom-up
- big bang

System Testing and Interoperability of OO Systems

Object oriented systems are by design meant to be built using smaller reusable components (i.e. the classes). Some of the reasons for this added importance are:
1. A class may have different parts, not all of which are used at the same time. When different clients start using a class, they may be using different parts of a class and this may introduce defects at a later (system testing) phase
2. Different classes may be combined together by a client and this combination may lead to new defects that are hitherto uncovered.
3. An instantiated object may not free all its allocated resource. Thus causing memory leaks and such related problems, which shows up only in the system testing phase

Regression Testing of OO Systems

- Taking the discussion of integration testing further, regression testing becomes very crucial for OO systems. As a result of the heavy reliance of OO systems on reusable components, changes to an one component could have potentially unintended side-effects on the clients that use the component.
- Hence, frequent integration and regression runs become very essential for testing OO systems. Also, because of the cascaded effects of changes resulting from properties like inheritance, it makes sense to catch the defects as early as possible.

Tools for Testing of OO Systems

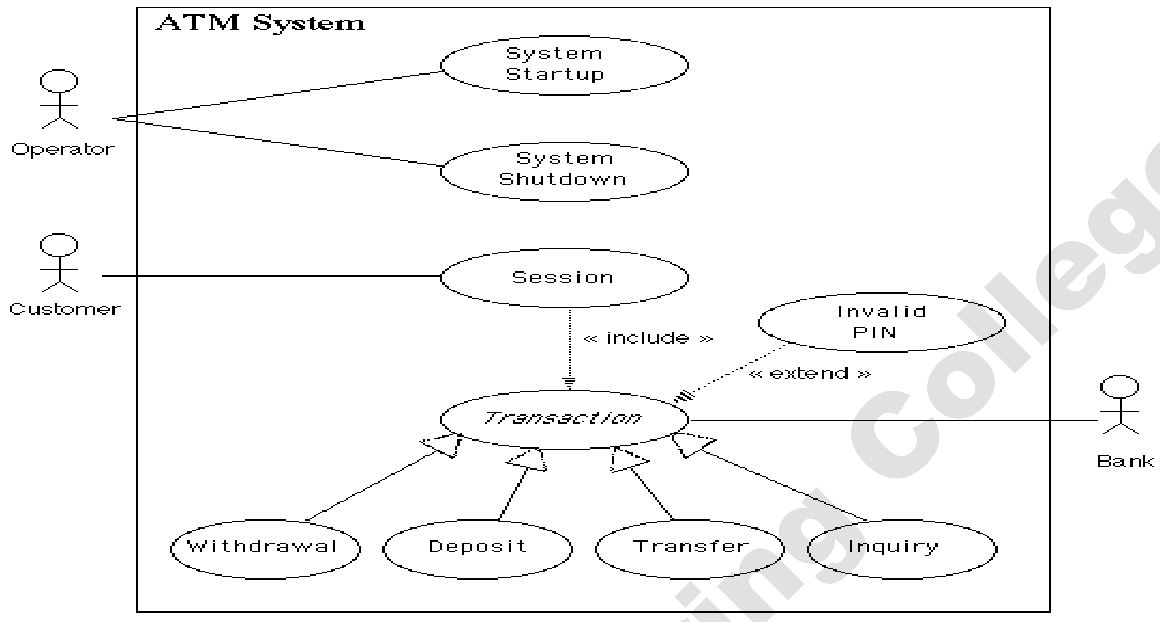
There are several tools that aid in testing OO systems. Some of these are

1. Use cases
2. Class diagrams
3. Sequence diagrams
4. Activity Diagrams
5.State charts

Arunai Engineering College

Use cases

Use cases represent the various tasks that a user will perform when interacting with the system. Use cases go into the details of the specific steps that the user will go through m accomplishing each task and the system responses for each steps. This fits in place for the object oriented paradigm, as the tasks and responses are akin to messages passed to the various objects.



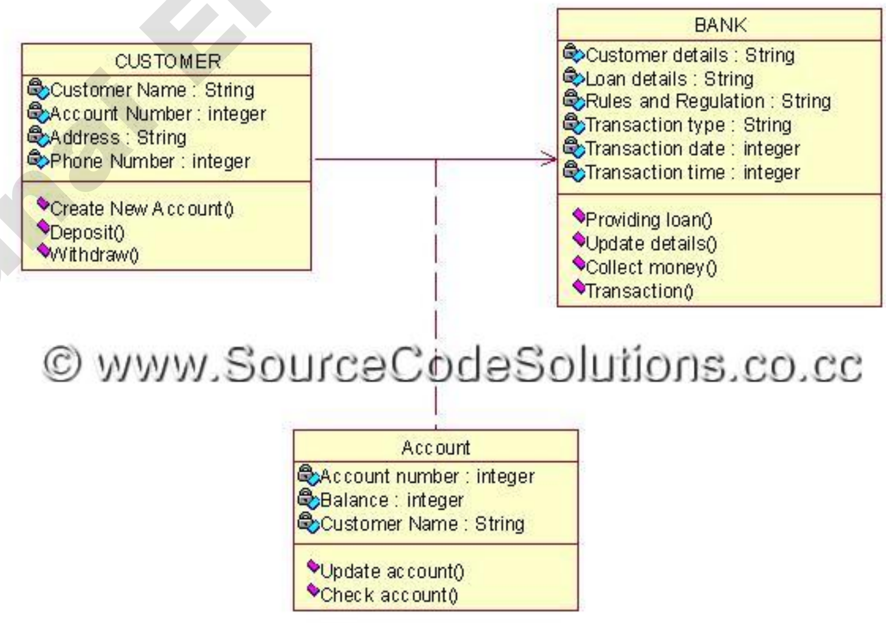
Class diagram

A class diagram is useful for testing in several ways.

1.It identifies the elements of a class and hence enables the identification of the boundary value analysis, equivalence partitioning, and such tests.

2.The associations help in identifying tests for referential integrity constraints across classes.

3.Generalizations help in identifying class hierarchies and thus help in planning incremental class testing as and when new variables and methods are introduced in child classes.



Arunai Engineering College

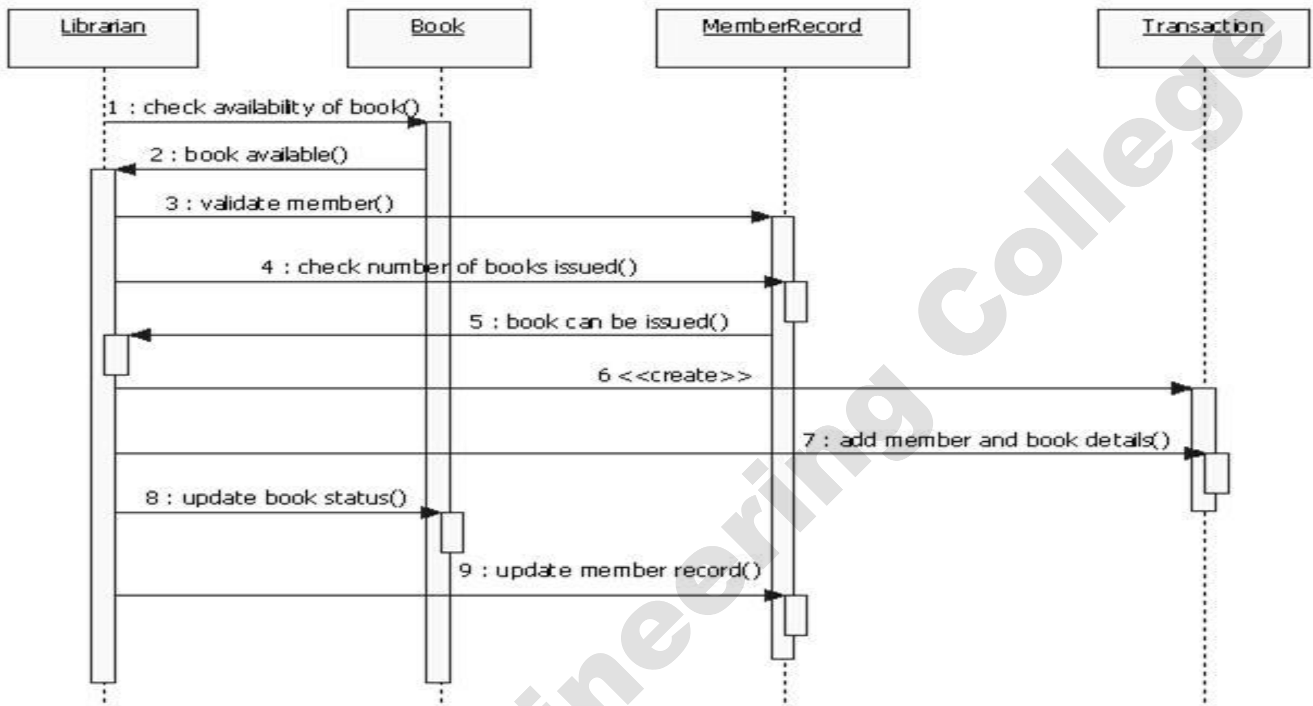
A sequence diagram

A sequence diagram helps in testing by

1. Identifying temporal end-to-end messages.
2. Tracing the intermediate points in an end-to-end transaction, thereby enabling easier narrowing down of problems.
3. Providing for several typical message-calling sequences like blocking call, non-blocking call, and so on.

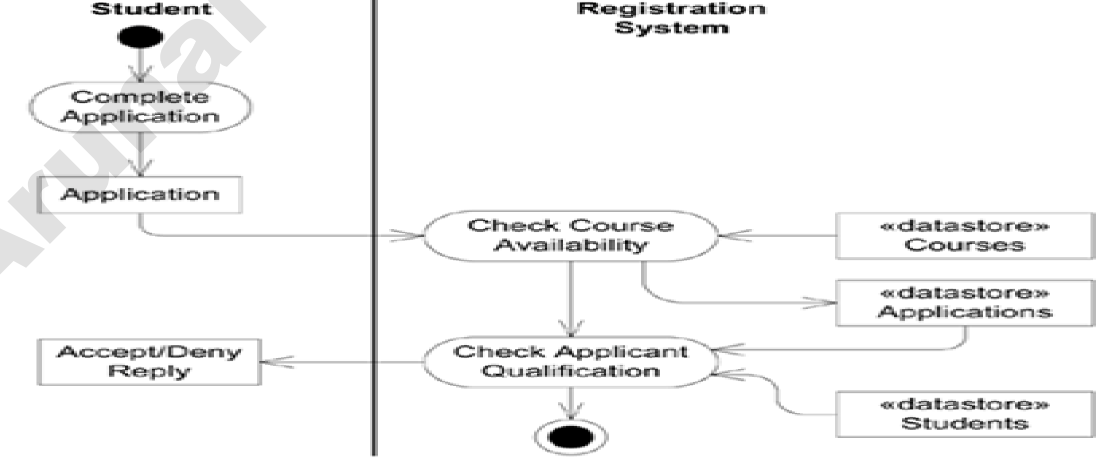
Sequence diagrams also have their limitations for testing-complex interactions become messy, if not impossible; to represent; dynamic binding cannot be represented easily.

Ex: Borrow Books in Library Information system



Activity diagram

While a sequence diagram looks at the sequence of messages, an activity diagram depicts the sequence of activities that take place. It is used for modeling a typical work flow in an application and brings out the elements of interaction between manual and automated processes. Since an activity diagram represents a sequence of activities, it is very similar to a flow chart and has parallels to most of the elements of a conventional flow chart.



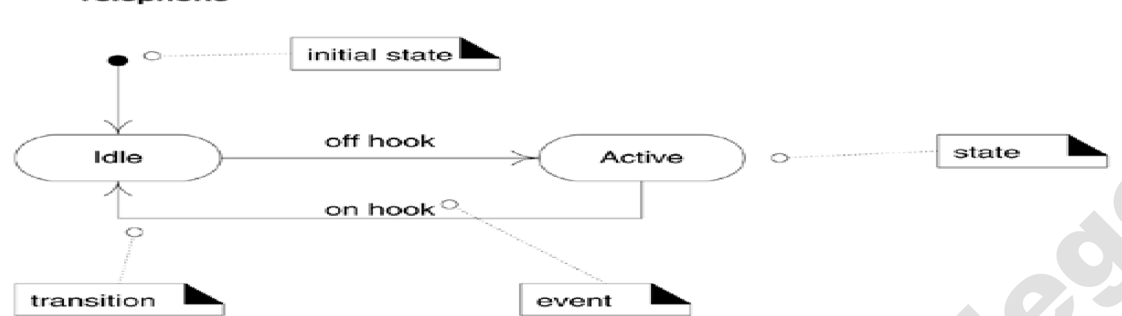
Given that an activity diagram represents control flow, its relevance for testing comes from

1. The ability to derive various paths through execution. Similar to the flow graph discussed in white box testing, an activity diagram can be used to arrive at the code complexity and independent paths through a program code.

2. Ability to identify the possible message flows between an activity and an object, thereby making the message-based testing more robust and effective.

State Chart Diagram

When an object can be modeled as a state machine, then the techniques of state-based testing, in black box testing can be directly applied.



USABILITY AND ACCESSIBILITY TESTING

Usability Testing

Testing that validates ease of use , speed and aesthetics of the product from the user's point of view

Characteristics

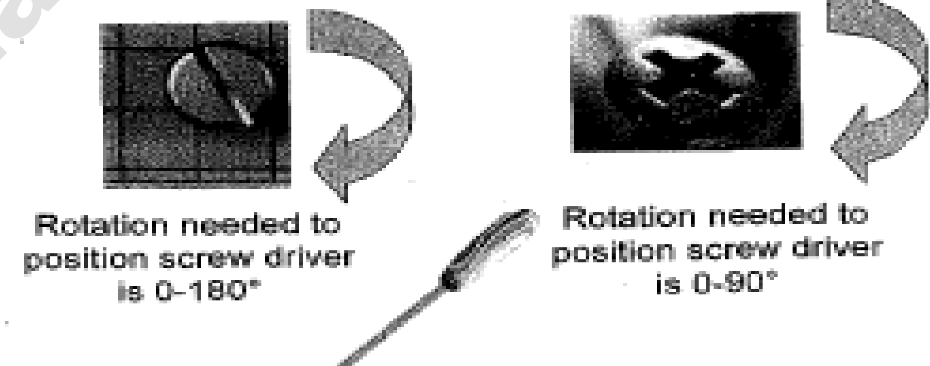
1. Usability testing tests the product from the users' point of view.
2. Usability testing is for checking the product to see if it is easy to use for the various categories of users.
3. Usability testing is a process to identify discrepancies between the user interface of the product and the human user requirements, in terms of the pleasantness and aesthetics aspects.

Conclusion

A view expressed by one user of the product may not be the view of another.

- easy for one user -->may not be easy for another
- fast (interms of say, response time) →e slow for another user
- beautiful by someone → look ugly to another.

APPROACH TO USABILITY



For example , when a Philips (or a star) screwdriver was invented, it saved only few milliseconds per operation to adjust the screwdriver to the angle of the screw compared to a flat screwdriver.

Arunai Engineering College

<u>People best suited to perform usability testing :</u>

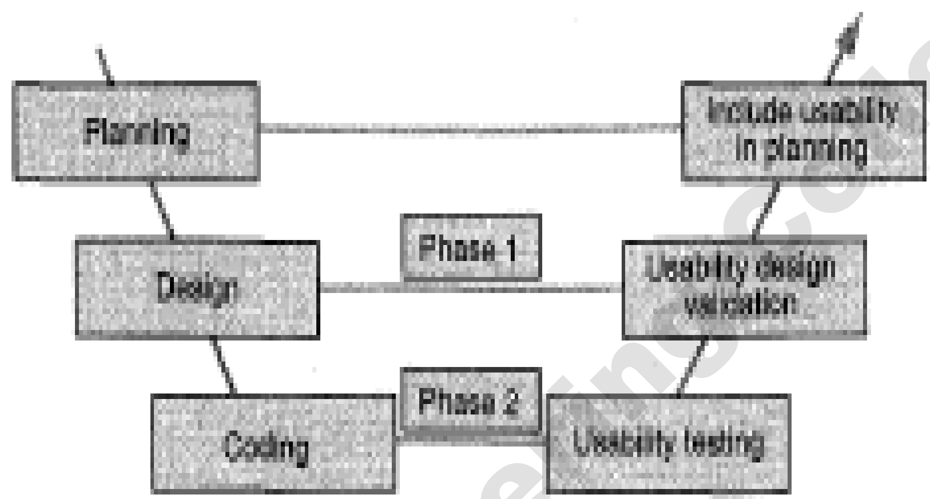
- representatives of the actual user segments who would be using the product
- People who are new to the product
-

<u>WHEN TO DO USABILITY TESTING?</u>

There are 2 phases in usability testing.

Phase 1 : Design Validation

Phase 2 : Usability testing done as a part of component and integration testing phases of a test cycle



Usability design is verified through several means, some of them are

- Style sheets
- Screen prototypes
- Paper designs
- Layout design

Web application interfaces are designed before designing functionality . That gives adequate time for doing two phases of usability testing.

<u>Client Application</u>	**<u>Web Application</u>**
<u>Step1 :</u> Design for functionality	**<u>Step1 :</u>** Design for User Interface
<u>Step2 :</u> Perform Coding for functionality	**<u>Step2 :</u>** Performa Coding for User Interface
<u>Step3 :</u> Design for User Interface	**<u>Step3 :</u>** Test User Interface (Phase 1)
<u>Step4 :</u> Perform coding for User Interface	**<u>Step4 :</u>** Design for Functionality
<u>Step5 :</u> Integrate user interface with functionality	**<u>Step5 :</u>** Perform coding for functionality
<u>Step6 :</u> Test UI along with functionality (Phase 1 & 2)	**<u>Step6 :</u>** Test UI along with functionality (Phase 2)

Development and Testing of Client Applications and Web Application

HOW TO ACHIEVE USABILITY?

Usability is a habit and a behavior, just like humans, the products are expected to behave differently and correctly with different users and to their expectations.

Checklists are created and verified during usability testing.

1. Do users complete the assigned tasks/operations successfully?
2. If so, how much time do they take to complete the tasks/operations?
3. Is the response from the product fast enough to satisfy them?
4. Where did the users get struck? What problems do they have?
5. Where do they get confused? Were they able to continue on their own? What helped them to continue?

QUALITY FACTORS FOR USABILITY

- **Comprehensibility** – when features and components are grouped in a product , they should be based on user terminologies not technology or implementation
- **Consistency** – A Product needs to be consistent with any applicable standards , platform look and feel , base infrastructure and earlier versions of the same product.
- **Navigation –** This helps in determining how easy it is to select the different operations of the product
- **Responsiveness-** How fast the product responds to the user request .

AESTHETICS TESTING

It ensures the product is pleasing to the eye.

Ex: A pleasant look for menus, pleasing colors, nice icons, and so on can improve aesthetics. It is generally considered as gold plating, which is not right.

ACCESSIBILITY TESTING

Verifying the product usability for physically challenged users

Accessibility to the product can be provided by two means.

1. Making use of accessibility features provided by the underlying infrastructure (for example, operating system), called basic accessibility, and
2. Providing accessibility in the product through standards and guidelines, called product accessibility.

I) Basic Accessibility

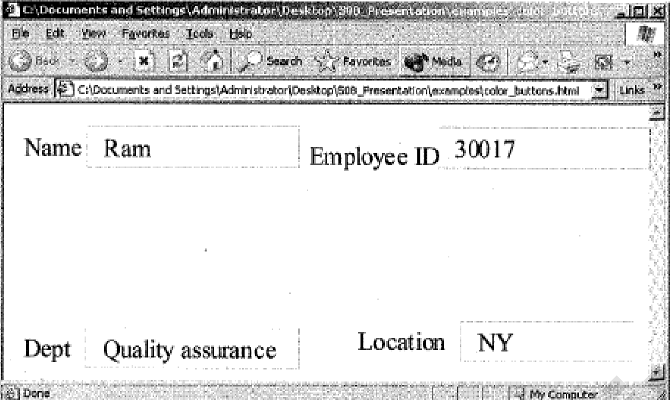
1) Keyboard accessibility
   - Sticky keys(ctrl, alt, del -->login , logout)
   - Filter keys
   - Toggle key sound
   - Sound keys
   - Arrow keys to control mouse
   - Narrator (text to audio)

Arunai Engineering College

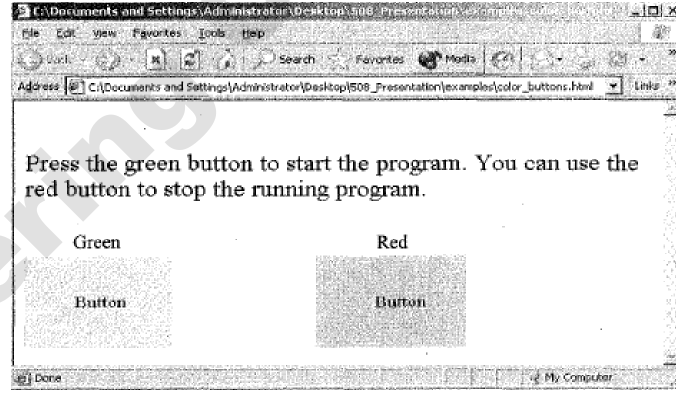
2) **Screen accessibility**
   - Visual sound
   - Enabling captions for multimedia
   - Soft keyboard
   - Easy reading with high contrast
3) **Other accessibility features**

II) Product Accessibility

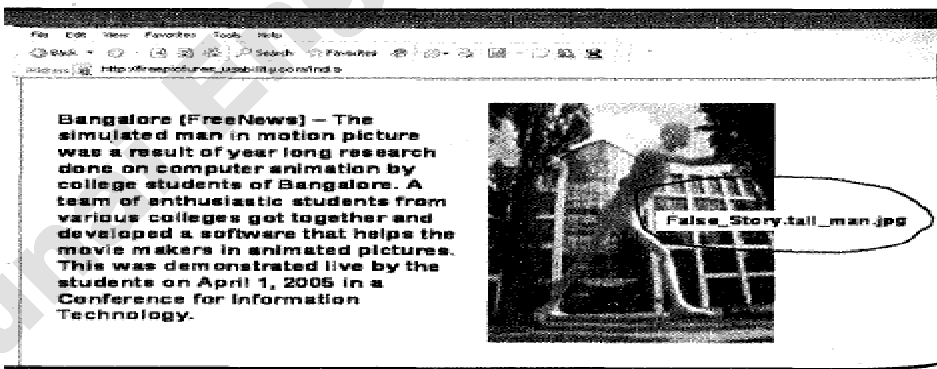
Sample Requirement 1	Text equivalent have to be provided for audio , video & picture images
Sample Requirement 2	Documents and fields should be organized (style sheets)
Sample Requirement 3	UI should be designed so that all info conveyed with color is also without color
Sample Requirement 4	Reduce the fictker rate , speed of moving text avoid flashes and blinking text
Sample Requirement 5	Reduce physical movements requirements for the user when designing the interface and allow adequate time for user response



Screen with 4 fields in the corner



Color as method of identification



Sample website with picture along with web site equivalent

TOOLS FOR USABILITY

- **Jaws**
- **HTML Validator**
- **Style Sheet Validator**
- **Magnifier (enlarge the items)**
- **Narrator (Text→ audio)**

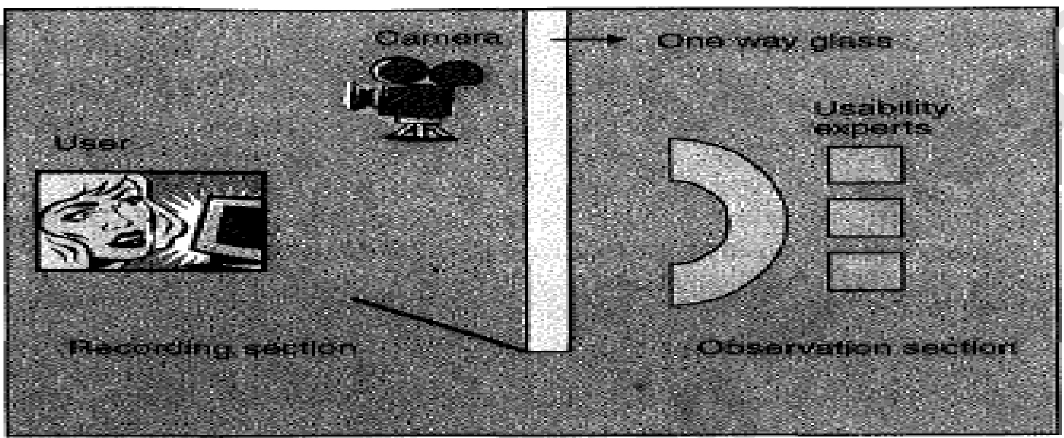
Arunai Engineering College

- **Soft Keyboard(display keyboard template on the screen)**

USABILITY LAB SETUP

This lab has 2 sections – recording sections and observation section.

- In the recording section of the lab - A user is requested to come to the lab with a prefixed set of operations that are to be performed with the product
- In the observations section of the lab - it is one way glass – the experts can see the user but the user cannot see the experts . some usability experts sit and observe the user for body language and associate the defects with the screens and events that caused it.



Configuration Testing (or refer page no 21 of unit 3 notes)

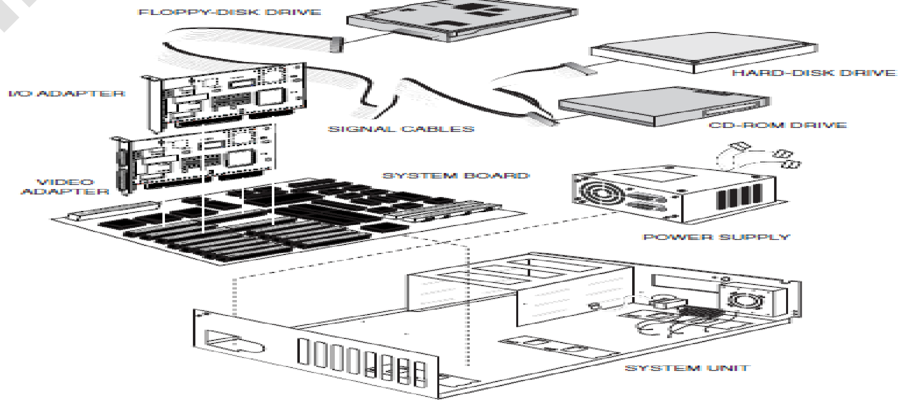
Configuration testing is the process of checking the operation of the software you're testing with all the various types of hardware.

Ex : Configuration bug

1. if your greeting card program works fine with laser printers but not with inkjet printers.
2. The hardware device or its device drivers may have a bug that only your software reveals. Maybe your software is the only one that uses a unique display card setting. When your software is run with a specific video card, the PC crashes.
3. if a specific printer driver always defaulted to draft mode and your photo printing software had to set it to high-quality every time it printed.

The PC.

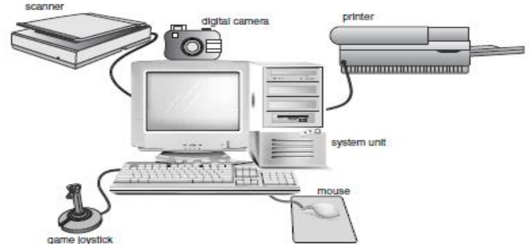
Components - system boards, component cards, and other internal devices such as disk drives, CD-ROM drives, video, sound, modem, and network cards



Peripherals. Peripherals, shown in Figure are the printers, scanners, mice, keyboards,

Arunai Engineering College

monitors, cameras, joysticks, and other devices that plug into your system and operate externally to the PC.



Interfaces. The components and peripherals plug into your PC through various types of
interface connectors .These interfaces can be internal or external to the PC. Typical names for them are ISA, PCI, USB, PS/2, RS/232, and Firewire. There are so many different possibilities that hardware manufacturers will often create the same
peripheral with different interfaces. It's possible to buy the exact same mouse in three
different configurations!

• **Options and memory.** Many components and peripherals can be purchased with different hardware options and memory sizes. Printers can be upgraded to support extra fonts or accept more memory to speed up printing. Graphics cards with more memory can support additional colors and higher resolutions.

• **Device Drivers.** All components and peripherals communicate with the operating system and the software applications through low-level software called device drivers. These drivers are often provided by the hardware device manufacturer and are installed when you set up the hardware. Although technically they are software, for testing purposes they are considered part of the hardware configuration.

configuration testing - the general process

1. **Decide the Types of Hardware You'll Need**
   Put your software disk on a table and ask yourself what hardware pieces you need to put together to make it work.

2. **Decide What Hardware Brands, Models, and Device Drivers Are Available**

Decide what device drivers you're going to test with. Your options are usually the drivers
included with the operating system, the drivers included with the device, or the latest drivers available on the hardware or operating system company's Web site.

3. **Decide Which Hardware Features, Modes, and Options Are Possible**

Color printers can print in black and white or color, they can print in different quality modes, and can have settings for printing photos or text. Display cards, as shown in Figure, can have different color settings and screen resolutions.

4. **Pare Down the Identified Hardware Configurations to a Manageable Set**

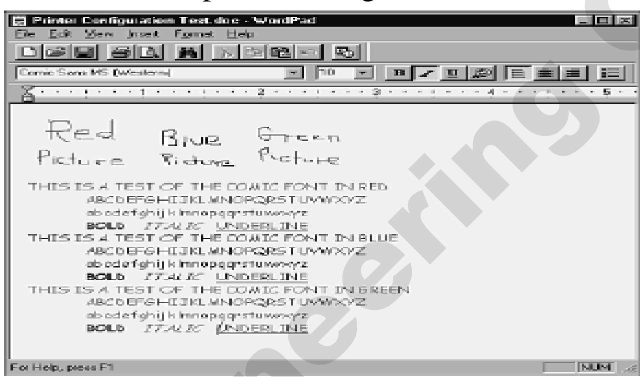
reduce the thousands of potential configurations into the ones that you're going to test.
put all the configuration information into a spreadsheet with columns for the manufacturer, model, driver versions, and options.

Arunai Engineering College

Popularity (1=most, 10=least)	Type (Laser / InkJet)	Age (years)	Manufacturer	Model	Device Driver Version	Options	Options
1	Laser	3	HAL Printers	LDIY2000	1.0	B/W	Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	1.0a	Color B/W	Draft Quality Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	2.0	Color B/W	Art Photo Draft Quality
10	Laser	5	OkeeDohKee	LJ100	1.5	B/W	100dpi 200dpi 300dpi
2	InkJet	2	OkeeDohKee	EasyPrint	1.0	Auto	600dpi

5. **Identify Your Software's Unique Features That Work with the Hardware Configurations**

For example, if you're testing a word processor such as WordPad), you don't need to test the file save and load feature in each configuration. File saving and loading has nothing to do with printing. A good test would be to create a document that contains different fonts, point sizes, colors, embedded pictures, and so on. You would then attempt to print this document on each chosen printer configuration



6. **Design the Test Cases to Run on Each Configuration**

    1. Select and set up the next test configuration from the list.
    2. Start the software.
    3. Load in the file configtest.doc.
    4. Confirm that the displayed file is correct.
    5. Print the document.
    6. Confirm that there are no error messages and that the printed document matches the standard.
    7. Log any discrepancies as a bug

7. **Execute the Tests on Each Configuration**

run the test cases and carefully log and report your results to your team, and to the ardware manufacturers if necessary. You'll need to work closely with the programmers and
white-box testers to isolate the cause and decide if the bugs you find are due to your software or to the hardware.
If the bug is specific to →the hardware, consult the manufacturer's Web site for information on reporting problems to them. Be sure to identify yourself as a software tester and what company you work for.

8. **Rerun the Tests Until the Results Satisfy Your Team**

It's difficult to run configuration testing the entire course of a project. Initially a few configurations might be tried, then a full test pass, then smaller and smaller sets to confirm bug fixes. Eventually you will get to a point where there

Arunai Engineering College

are no known bugs or to where the bugs that still exist are in uncommon or unlikely test configurations. At that point, you can call your configuration testing complete.

Compatibility Testing (Refer Unit 2 Notes)
Documentation Testing (Refer Unit 2 Notes)

Website Testing

- Web Page Fundamentals
- Black-Box Testing
- Gray-Box Testing
- White-Box Testing
- Configuration and Compatibility Testing
  - Usability Testing

Web Page Fundamentals

Internet Web pages are just documents of text, pictures, sounds, video, and hyperlinks
Web page features.

- Text of different sizes, fonts, and colors (okay, you can't see the colors in this book)
- Graphics and photos
- Hyperlinked text and graphics
- Varying advertisements
- Drop-down selection boxes
- Fields in which the users can enter data

features that make the Web site much more complex:

- Customizable layout that allows users to change where information is positioned
- onscreen
- Customizable content that allows users to select what news and information they want to see
- Dynamic drop-down selection boxes
- Dynamically changing text
- Dynamic layout and optional information based on screen resolution
- Compatibility with different Web browsers, browser versions, and hardware and software platforms
- Lots of hidden formatting, tagging, and embedded information that enhances the Web page's usability

Testing Techniques apply to Web page testing

- basic white-box and black-box techniques
- configuration and compatibility testing
- usability testing

1) **Black-Box Testing**

screen image of Apple's Web site, www.apple.com, a fairly straightforward and typical Web site. It has all the basic elements—text, graphics, hyperlinks to other pages on the site, and hyperlinks to other Web sites.

Arunai Engineering College

The easiest place to start is by treating the Web page or the entire Web site as a black box
What would you test? What would you choose not to test?

When testing a Web site, you first should create a state table , treating each page as a different state with the hyperlinks as the lines connecting them. A completed state map will give you a better view of the overall task.

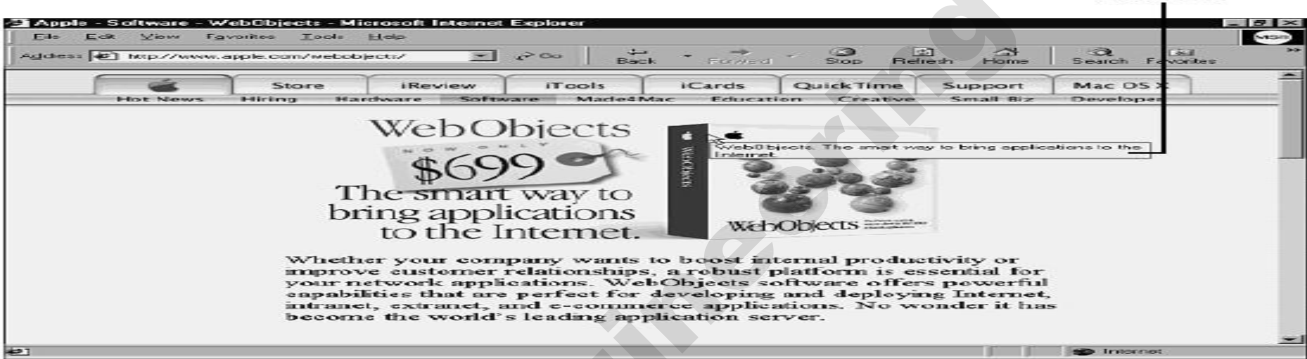
Web pages are made up of just text, graphics, links, and the occasional form. Testing them isn't difficult.

Text

Check the audience level,

- the terminology,
- the content and subject matter,
- the accuracy—especially of information that can become outdated—and
- always check spelling.
- each page has a correct title

An often overlooked type of text is called ALT text, for ALTernate text. Figure shows an example of ALT text. When a user puts the mouse cursor over a graphic on the page he gets a pop-up description of what the graphic represents. Web browsers that don't display graphics use ALT text. Also, with ALT text blind users can use graphically rich Web sites—an audible reader interprets the ALT text and reads it out through the computer's speakers.



Hyperlinks

Links can be tied to text or graphics. Each link should be checked to make sure that it jumps to the correct destination and opens in the correct window.

Check

- Text links are usually underlined, and the mouse pointer should change to a hand pointer when it's over any kind of hyperlink—text or graphic.
- Look for orphan pages, which are part of the Web site but can't be accessed through a hyperlink
- do all graphics load and display properly? If a graphic is missing or is incorrectly named, it won't load and the Web page will display an error where the graphic was to be placed.
- If text and graphics are intermixed on the page, make sure that the text wraps properly around the graphics. Try resizing the browser's window to see if strange wrapping occurs around the graphic.
- How's the performance of loading the page? Are there so many graphics on the page, resulting in a large amount of data to be transferred and displayed, that the Web site's performance is too slow?
- What if it's displayed over a slow dial-up modem connection on a poor-quality phone line?

If a graphic can't load onto a Web page, an error box is put in its location

Arunai Engineering College

Forms

Forms are the text boxes, list boxes, and other fields for entering or selecting information on a Web page. In the example a signup form for potential Mac developers. There are fields for entering your first name, middle initial, last name, and email address.

Make sure your Web site's form fields are positioned properly. Notice in this Apple Developer signup form that the middle initial (M.I.) field is misplaced.

Gray-Box Testing

graybox testing, is a mixture of the black box & white box testing —hence the name. You still test the software as a black-box, but you supplement the work by taking a peek (not a full look, as in white-box testing) at what makes the software work. Web pages provide themselves nicely to gray-box testing.

Most Web pages are built with HTML (Hypertext Markup Language). Listing shows a few lines of the HTML used to create the Web page

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
….

HTML and Web pages can be tested as a gray box because HTML isn't a programming language it's a markup language.

In the early days of word processors, you couldn't just select text and make it bold or italic. You had to embed markups, sometimes called field tags, in the text. For example, to create the bolded phrase

This is bold text.

you would enter something such as this into your word processor:

[begin bold]This is bold text.[end bold]

HTML works the same way. To create the line in HTML you would enter

This is bold text.

HTML has evolved to where it now has hundreds of different field tags and options, as evidenced by the HTML

2) White-Box Testing

Web page also has customizable and dynamic changing content. Remember, HTML isn't a programming language—it's merely a tagging system for text and graphics. To create these extra dynamic features requires the HTML to be

Arunai Engineering College

supplemented with programming code that can execute and follow decision paths. popular Web programming languages: DHTML, Java, JavaScript, ActiveX, VBScript, Perl, CGI, ASP, and XML.

the important bugs that you have some knowledge of the Web site's system structure and programming:
• **Dynamic Content.** Dynamic content is graphics and text that changes based on certain conditions—for example, the time of day, the user's preferences, or specific user actions.
Supported by

- Client side scripting :It's possible that the programming for the content is done in a simple scripting language such as JavaScript and is embedded within the HTML. apply gray-box testing techniques when you examine the script and view the HTML.

- server-side scripting : For efficiency, most dynamic content programming is located on the Web site's server; and would require to have access to the Web server to view the code.

• **Database-Driven Web Pages.** Many e-commerce Web pages that show catalogs or inventories are database driven. The HTML provides a simple layout for the Web content and then pictures, text descriptions, pricing information, and so on are pulled from a database on the Web site's server and plugged into the pages.

• **Programmatically Created Web Pages.** Many Web pages, especially ones with dynamic content, are programmatically generated—that is, the HTML and possibly even the programming is created by software. A Web page designer may type entries in a database and drag and drop elements in a layout program, press a button, and out comes the HTML that displays a Web page. If you're testing such a system, you have to check that the HTML it creates is what the designer expects.

• **Server Performance and Loading**. Popular Web sites might receive millions of individual hits a day. Each one requires a download of data from the Web site's server to the browser's computer. If you wanted to test a system for performance and loading, you'd have to find a way to simulate the millions of connections and downloads.

• **Security**. Web site security issues are always in the news as hackers try new and different ways to gain access to a Web site's internal data. Financial, medical, and other Web sites that contain personal data are especially at risk and require intimate knowledge of server technology to test them for proper security.

3)Configuration and Compatibility Testing

Configuration testing is the process of checking the operation of your software with various types of hardware and software platforms and their different settings.
Compatibility testing is checking your software's operation with other software.
Web pages are perfect examples of where you can apply this type of testing .Assume that you have a Web site to test. You need to think about what the possible hardware and software configurations might be that could affect the operation or appearance of the site.
Here's a list to consider:
• **Hardware Platform.** Is it a Mac, PC, a TV browsing device, a hand-held, or a wristwatch? Each hardware device has its own operating system, screen layout, communications software, and so on. Each can affect how the Web site appears onscreen.
• **Browser Software and Version.** There are many different Web browsers and browser

Arunai Engineering College

versions. Some run on only one type of hardware platform, others run on multiple platforms. Some examples are Netscape Navigator 3.04 and 4.05, Internet Explorer 3.02,
4.01, and 5.0, Mosaic 3.0, Opera, and Emacs.

• **Browser Plug-Ins.** Many browsers can accept plug-ins or extensions to gain additional
functionality. An example of this would be to play specific types of audio or video files.

• **Browser Options.** Most Web browsers allow for a great deal of customization. You can select security options, choose how ALT text is handled, decide what plug-ins to enable, and so on. Each option has potential impact on how your Web site operates—and, hence, is a test scenario to consider.

Video Resolution and Color Depth. Many platforms can display in various screen resolutions and colors. A PC running Windows, for example, can have screen dimensions of
640×480, 800×600, 1,024×768, 1280×1024, and up. Your Web site may look different, or
even wrong, in one resolution, but not in another. Text and graphics can wrap differently,
be cut off, or not appear at all. The number of colors that the platform supports can also impact the look of your site. There can be as few as 16 colors and as many as 224. Could your Web site be used on a system with only 16 colors?

• **Text Size.** Did you know that a user can change the size of the text used in the browser?
Could your site be used with very small or very large text? What if it was being run on a
small screen, in a low resolution, with large text?

• **Modem Speeds.** Enough can't be said about performance. Someday everyone will have
high-speed connections with Web site data delivered as fast as you can view it. Until
then, you need to test that your Web site works well at a wide range of modem speeds.

4) Usability Testing

The following list is adapted from his Top Ten Mistakes in Web Design:

• **Gratuitous Use of Bleeding-Edge Technology**. Your Web site shouldn't try to attract
users by bragging about its use of the latest Web technology. When desktop publishing was young, people put 20 different fonts in their documents; try to avoid similar design bloat on the Web.

• **Scrolling Text, Marquees, and Constantly Running Animations**. Never allow page
elements that move incessantly. Moving images have an overpowering effect on human
peripheral vision.

• **Long Scrolling Pages.** Users typically don't like to scroll beyond the information visible
onscreen when a page comes up. All critical content and navigation options should be on
the top part of the page. Recent studies have shown that users are becoming more willing

Arunai Engineering College

to scroll now than they were in the early years of the Web, but it's still a good idea to
minimize scrolling on navigation pages.

• **Non-Standard Link Colors**. Hyperlinks to pages that users haven't seen should be blue;
links to previously seen pages should be purple or red. Don't mess with these colors
because the ability to understand which links have been followed is one of the few navigational aids that's standard in most Web browsers. Consistency is key to teaching users what the link colors mean.

• **Outdated Information.** some pages are better off being removed completely from the server after their expiration date.

• **Overly Long Download Times.** Traditional human-factor guidelines indicate that 0.1 second is about the limit for users to feel that the system is reacting instantaneously. One second is about the limit for a user's flow of thought to stay uninterrupted. Ten seconds is the maximum response time before a user loses interest. On the Web, users have been trained to endure so much suffering that it may be acceptable to increase this limit to 15 seconds for a few pages. But don't aim for this—aim for less.

• **Lack of Navigation Support**. They will always have difficulty finding information, so they need support in the form of a strong sense of structure and place. Your site's design should start with a good understanding of the structure of the information space and communicate that structure explicitly to users. Provide a site map to let users know where they are and where they can go. The site should also have a good search feature because even the best navigation support will never be enough.

• **Orphan Pages**. Make sure that all pages include a clear indication of what Web site they
belong to since users may access pages directly without coming in through your home page. For the same reason, every page should have a link to your home page as well as some indication of where they fit within the structure of your information space.

• **Complex Web Site Addresses (URLs).** Even though machine-level addressing like the URL should never have been exposed in the user interface, it's there and research has found that users actually try to decode the URLs of pages to infer the structure of Web sites. Users do this because of the lack of support for navigation and sense of location in current Web browsers. Thus, a URL should contain human-readable names that reflect the nature of the Web site's contents.

• **Using Frames**. Frames are an HTML technology that allows a Web site to display another Web site within itself, hence the name frame—like a picture frame. Splitting a page into frames can confuse users since frames break the fundamental user model of the Web page.

Arunai Engineering College

UNIT IV TEST MANAGEMENT

People and organizational issues in testing – Organization structures for testing teams – testing services – Test Planning – Test Plan Components – Test Plan Attachments – Locating Test Items – test management – test process – Reporting Test Results – Introducing the test specialist – Skills needed by a test specialist – Building a Testing Group-The structure of testing group-The Technical training program.

PEOPLE AND ORGANIZATIONAL ISSUES IS TESTING

COMMON PEOPLE ISSUES

Perceptions and Misconceptions about testing

- ***Testing is not technically challenging***
  - Requires a holistic understanding of the entire product
  - Requires thorough understanding of multiple domains
  - Specialization in languages, Use of tools
  - Opportunities for conceptualization and out-of-the-box thinking.
  - Significant investments are made in testing today – sometimes a lot more than in development
- ***Testing does not provide me a career path or growth***
  - Normally job titles are given as "Development Engineers", "Senior Development Engineer" etc.
  - Testing organizations do not always present such obvious opportunities for career growth. This does not mean that there are no career paths for testing professionals.
  - There is an equally lucrative career path for testing professionals also.
- ***I am put in testing – what is wrong with me?!***
  - Toppers allocated for development and testing functions get the leftovers, obviously management is sending the wrong signals and reinforcing the wrong message.
  - A person assigned testing only when he or she has the right aptitude and attitude for testing.
  - Compensations and reward favor the development leads to people "graduate to development" rather than look for careers in testing itself
- ***These folks are my adversaries***
  - Testing and development teams should reinforce each other and not be at loggerheads.
- ***Testing is what I can do in the end if I get time***
  - Testing is not what happens in the end of a project – it happens throughout and

Arunai Engineering College

continues even beyond a release

- ***There is no sense of ownership in testing***
  - Testing has deliverables just as development has and hence testers should have the same sense of ownership
- ***Testing is only destructive***
  - Testing is destructive as much it is constructive, like the two sides of a coin.

Providing Career Path for Testing Professional

When people look for a career path in testing, some of the areas of progression they look for are,

Arunai Engineering College

- Technical challenge
- Learning opportunities
- Increasing responsibilities and authority
- Increasing independence
- Ability to have a significant influence
- Rewards and recognition

Responsibilities of a Test Engineer

- Following the test process for executing tests, maintaining tests etc.
- Filing high quality defects, usable by developers
- Categorizing defects
- Adhering to schedules specified
- Developing high quality documentation

Responsibilities of a Senior Test Engineer

- Helping development staff in debugging and problem isolation
- Contribution to enhancing processes for testing
- Generation of metrics related to testing

Responsibilities of a Test Lead

- Review of test case, test design etc.
- Planning test strategy
- Allocating tasks to individual and monitoring it
- Mentoring team members and assisting them in technical matters
- Interaction with developing team for debugging and problem reproduction
- Overall responsibility for test quality

Responsibilities of a Test Architect

- A test architect has in-depth knowledge of a variety of testing techniques and methodologies used both inside and outside of an organization
- They often provide technical assistance and/or advice to the test Manager.
- Test Architect come into picture when Test Manager takes on additional responsibilities
- A test architect is expected to be able to affect change not only across the testing community, but between other engineering disciplines as well.

Development Functions Vs Testing

Similarities

1. **Requirement / Test Specification** : Requires thorough understanding of the domain.
2. **Design** : Carries with it all attributes for product design like reuse, standard formation.
3. **Coding / Test Script** : Involves using the development and test automation tools.
4. **Testing / Making the tests operational** : Involves well-knot teamwork between teams to ensure that correct results are captured.
5. **Maintenance** : Keeping the tests current with changes from maintenance.

Arunai Engineering College

Differences

1. Testing is often a crunch time function
   Testing functions close to product release time throws in some unique planning and management challenge.
2. More "elasticity" is allowed in projects in earlier phases
   Development function will take longer than planned, whereas same amount is not given for testing as final deadline for a product release is seldom compromised.
3. Testing is arguably the most difficult ones to staff
   It is difficult to attract and retain top talent for testing functions.
4. Testing usually carry more external dependencies that development functions.

The role of the ecosystem and call for action

- ✓ ***Role of education system***: The right values can only be more effectively **caught** by the students than be **taught** by the teachers.
  - Not been as core course; No lab experience; No real time experience
  - Projects not asking test plan, but for coding only
  - Scope for team work???
- ✓ ***Role of senior management***: Fairness to and recognition of testing professionals should not only be **done** but should be **seen to** be done.
  - How to spot a good tester.
  - Rewarding people.
- ✓ ***Role of the community***: As members of test community, do you have pride and sense of equality/ Remember, authority is **taken, not given**.
  - Pride in work

ORGANISATION STRUCTURES FOR TESTING TEAMS

ORGANIZATION STRUCTURE FOR TESTING TEAMS:

Dimensions of Organization Structures

The organization structures are based on two dimensions.

1. Dimension based on organization type.
2. Dimension based on geographic distribution

The organization is classified into two types. They are:

1. Product Organization – Product Organization produces software products and responsible for the entire product.
2. Service Organization – Service organization does not have complete

Arunai Engineering College

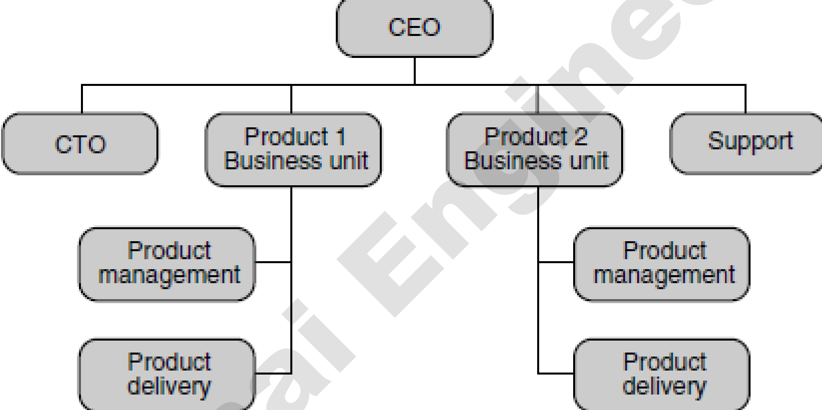
product responsibility.

Dimensions of Organization Structure

- **Organization Type**
- **Geographic Distribution**

  - Organization types
    - **Product**
      - Responsibility for entire product
      - Testing is one phase
    - **Service**
      - Provide testing service to other organizations
      - Provide test specialist
  - Geographic distribution
    - Single site → all members at one place
    - Multi site → members at different location

STRUCTURES IN SINGLE-PRODUCT COMPANIES
Product companies in general have a high-level organization structure similar to the one shown in the figure shown below:



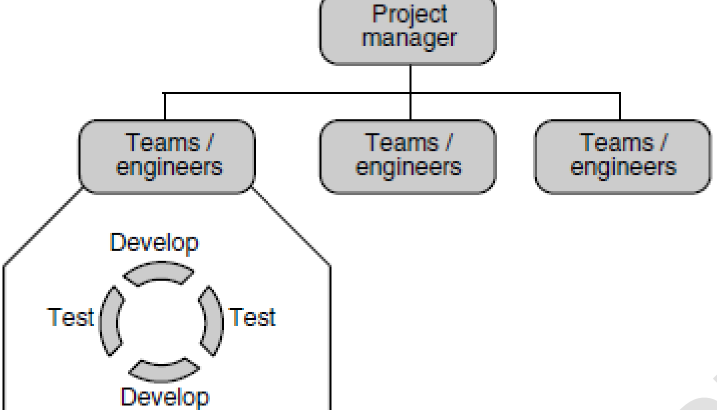
Organization structure of a multi-product company.

Arunai Engineering College

Testing Team Structure for Single-Product Companies

- Most product companies start with a single product.
- During the initial stages of evolution, the organization does not work with many formalized processes.
- The product delivery team members distribute their time among multiple tasks and after wear multiple hats.

All the engineers report into the project manager who is in charge of the entire project, with very little distinction between testing function and development functions. Thus, there is only a very thin line separating the ―development team and ―testing team.



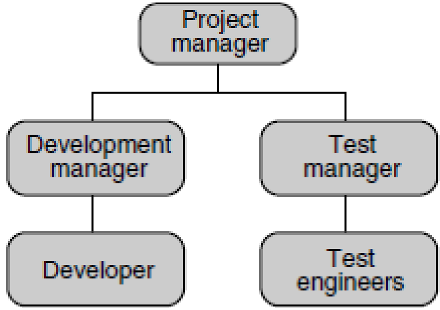
Typical organization structures in early stages of a product.

- **Advantages**
  - ❖ Exploits the rear-loading nature of testing activities (during early part of the project, everyone chips in for development and during the later part, they switch over to test)
  - ❖ Enables engineers to gain experience in all aspects of life cycle.
  - ❖ Is amenable to the fact that the organization mostly only has informal processes.
  - ❖ Some defects may be detected early.

- **Disadvantages**
  - ❖ Accountability for testing and quality reduces (give importance for developing or testing?)
  - ❖ Developers do not in general like testing and hence the effectiveness of testing suffers
  - ❖ Schedule pressures generally compromise testing (deadline to meet leads to insufficient time for testing causes compromise in quality of testing).
  - ❖ Developers may not be able carry out the different types of tests.

Separate groups for testing and development.

Arunai Engineering College

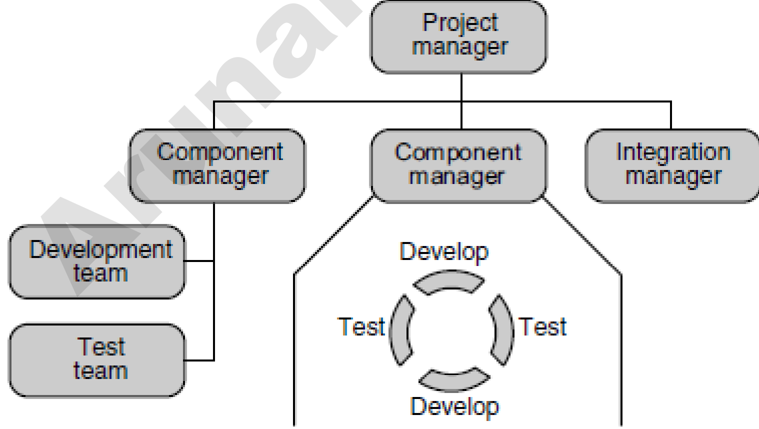


1.There is clear accountability for testing and development. The results and the expectations from the two teams can be more clearly set and demarcated.

2. Testing provides an external perspective. Since the testing and development teams are logically separated, there is not likely to be as much bias as in the previous case for the testers to prove that the product works. This external perspective can lead to uncovering more defects in the product.

3. Takes into account the different skill sets required for testing. As we have seen in the earlier chapters, the skill sets required for testing functions are quite different from that required for development functions. This model recognizes the difference in skill sets and proactively address the same.

Component-Wise Testing Teams: Even if a company produces only one product, the product is made up of a number of components that fit together as a whole. In order to provide better accountability, each component may be developed and tested by separate teams and all the components integrated by a single integration test team reporting to the project manager.



COMPONENT WISE ORGANIZATION

Arunai Engineering College

Testing Team Structure for Multi-Product Companies

The organization of test teams in multi-product companies is dictated largely by the following factors.

- How tightly couples the products are in terms of technology
- Dependence among various products
- How synchronous are the release cycles of products
- Customer base for each product & similarity among customer bases for various products.

Arunai Engineering College

Based on the above factors, there are several options available for organizing testing teams.

1. Testing team as part of "CTO's Office"
    - Developing a product architecture that is testable or suitable for testing
    - Testing team will have better product and technology skills
    - The testing team can get a clear understating of what design and architecture are built for and plan their tests accordingly
    - The CTO's team can evolve a consistent, cost-effective strategy for test automation.
    - In order to make the test team more effective,
        - It should be smaller in number
        - It should be a team of equals or at most very few hierarchies.
        - It should have organization-wide representation
        - It should have decision-making and enforcing authority
        - It should be involved in periodic reviews to ensure operations are in line.
2. Single test team for all products.

    This model is similar to the case of a single product team divided into multiple components and each of the components being developed by an independent team. Here, since different groups have delivery responsibilities for different products, the single testing team must report to a different level. The possibilities are,

    (i) The single testing team can form a "testing business unit" and report into this unit.
    (ii) The testing team can be made to report to the "CTO thing-tank".
3. Testing teams organized by product

    The issues in single testing teams are, accountability, decision making and scheduling. Solution?...

    Assign complete responsibility of all aspects of a product to the corresponding business unit and let the business unit head figure out how to organize the testing and development teams.
4. Separate testing team for different phases of testing

Types of test	**Reports into**
White box testing	Development team
Black box testing	Testing team
Integration testing	Organization-wide testing team
System testing	Product management / Product marketing
Performance testing	A central benchmarking group
Acceptance testing	Product management / Product marketing
Internationalization testing	internationalization team and some local teams
Regression testing	All test Teams

Advantages

1. People with appropriate skill sets are used to perform a given type of test.
2. Defects can get detected better and closer to the point of injection.

Arunai Engineering College

TESTING SERVICES – (SERVICE STRUCTURE OF TESTING TEAM)

Testing Services Organization

Today it is common to find testing activities outsourced to external companies who specialize is testing and provide testing services.

Business Need for Testing Services

1. Testing is becoming increasingly diverse and a very specialized function
2. The variety and complexity of the test automation tools further increase the challenge in testing.
3. Testing as a process is becoming better defined and thus makes testing to outsourcing.
4. An organization expertise in understanding software domain may be expert in setting up and running an effective testing function.
5. An outsourced organization can offer location and cost advantage.

Typical Roles and Responsibilities of Testing Services Organization

- ➢ A testing services organization is made up of a number of accounts.
- ➢ Each account being responsible for a major customer.
- ➢ Each account is assigned an account manager.
- ➢ ***Account manager***, a single point of contact from the customer into testing services organization.

  Account manager service:
  - Single point contact between customer and testing service organization.
  - Develops rapport with customer; responsible for ensuring current projects are delivered as promised; getting new business.
  - Participate in all strategic (and tactical) communication between them.
  - Acts as a proxy for the customer within the testing services organization.

- ✓ Account manager may be ***located*** close to the customer site or at the location of testing services organization.
- ✓ To develop better rapport, this role would ***require frequent travel and face-to-face meetings*** with the customer.

The testing service team organizes its account team as a ***near-site team*** and a ***remote team***.

Near-site team → Usually a small team, placed at or near customer location.

- Direct first point of contact for the costumer for tactical and urgent issues
- Act as a stop-gap to represent the remote team, in the event of emergencies.
- Serves to increase the rapport between the customer's operational team and the testing services team.

Arunai Engineering College

Remote team → Usually large in number the team does the bulk of work, located on the site of the testing services organization.

- The ***remote team manager*** manages the entire remote team.
- Can have a peer-to-peer relationship with the near-site team or have the near-site team reporting to them.
- Can have further hierarchies as test leads and test engineers.

Challenges and Issues in Testing Service Organizations

1. The outsider effect and estimation of resources
   - Do not necessarily have access to the product internals or code.
   - Do not have access to product history (to find which modules has historically problem prone – results been in not able to prioritize testing).
   - May not necessarily have same level of rapport with development team.
   - Will have to estimate and plan for hardware and software resources.
2. Domain expertise
   - Testing service organization may have to undertake projects from multiple customers.
   - The little product ownership makes it tougher to get domain expertise to join a testing service company.
   - The diversity of domains exacerbates the problem.
3. Privacy and customer isolation issues
   - As testing service organization has a common infrastructure, physical isolation of the different teams may be difficult
   - As people more from one project to another, there should be full confidence and transparency in ensuring the customer-specific knowledge acquired in one project is not taken to other projects.
   - A Non Disclosure Agreement (NDA) is draw up between the customer and the testing service organization.
4. Apportioning hardware and software resources and costs
   - When dealing with multiple customers, the organization uses hardware and software resources internally.
   - Some of them are identified and allocated for particular account and some others and multiplexed across multiple projects (eg. Satellite links, e-mail server etc.).
   - The cost has to be apportioned across different projects while costing the project.
5. Maintaining the "bench"
   - Need to maintain "people of bench" – people not allocated to any project but ready in the wings to take on new projects and to convince customers.
     - New projects may come at any time
     - Some prospects may require initial studies to be done or some demonstration of initial capability before signing.
     - May need to know resumes of specific individuals and to select people at initial level.

Arunai Engineering College

Success Factors for Testing Organizations

(i) Communication and teamwork
(ii) Bringing in customer perspective
(iii) Providing appropriate tools and environment
(iv) Providing periodic skill upgrades

TEST PLANNING

A plan is a document that provides a framework or approach for achieving a set of goals.

In order to meet a set of goals, a plan describes what specific tasks must be accomplished, who is responsible for each task, what tools, procedures, and techniques must be used, how much time and effort is needed, and what resources are essential. A plan also contains milestones.

Milestones are tangible events that are expected to occur at a certain time in the project's lifetime. Managers use them to determine project status.

Tracking the actual occurrence of the milestone events allows a manager to determine if the project is progressing as planned. Finally, a plan should assess the risks involved in carrying out the project.

4.0 Test Planning

4.1 Meet with project manager. Discuss test requirements.
4.2 Meet with SQA group, client group. Discuss quality goals and plans.
4.3 Identify constraints and risks of testing.
4.4 Develop goals and objectives for testing. Define scope.
4.5 Select test team.
4.6 Decide on training required.
4.7 Meet with test team to discuss test strategies, test approach, test monitoring, and controlling mechanisms.
4.8 Develop the test plan document.
4.9 Develop test plan attachments (test cases, test procedures, test scripts).
4.10 Assign roles and responsibilities.
4.11 Meet with SQA, project manager, test team, and clients to review test plan.

Test plans for software projects are very complex and detailed documents. The planner usually includes the following essential high-level items.

1. ***Overall test objectives***. As testers, why are we testing, what is to be achieved by the tests, and what are the risks associated with testing this product?

Arunai Engineering College

2. *What to test* *(scope of the tests).* What items, features, procedures, functions, objects, clusters, and subsystems will be tested?
3. *Who will test*. Who are the personnel responsible for the tests?
4. *How to test*. What strategies, methods, hardware, software tools, and techniques are going to be applied? What test documents and deliverable should be produced?
5. *When to test*. What are the schedules for tests? What items need to be available?
6. *When to stop testing*. It is not economically feasible or practical to plan to test until all defects have been revealed. This is a goal that testers can never be sure they have reached. Because of budgets, scheduling, customer deadlines, specific conditions must be outlined in the test plan.

Test plans can be organized in several ways depending on organizational policy. The complexity of the hierarchy depends on the type, size, risk-proneness, and the mission/safety criticality of software system being developed. All of the quality and testing plans should also be coordinated with the overall software project plan. A sample plan hierarchy is shown in following figure.

Software quality assurance plan → This plan gives an overview of all verification and validation activities for the project, details related to other quality issues such as audits, standards, configuration control, and supplier control.

Master test plan → An overall description of all execution-based testing for the software system.
Master verification plan → For reviews inspections/walkthroughs The master test plan itself may be a component of the overall project plan or exist as a separate test plan for unit, integration, system, and acceptance tests.
The level-based plans give a more detailed view of testing appropriate to that level.

The **persons responsible** for developing test plans depend on the type of plan under development. For example, the **master test plan** for execution-based testing may be developed by the **project manager**, especially if there is no separate testing group. A **tester or software quality assurance manager**, can also do this but always requires cooperation and input from the project manager.

The type and organization of the test plan, the test plan hierarchy, and who is responsible for development should be specified in organizational standards or SQA documents.

TEST PLAN COMPONENTS

The basic test plan components as described in IEEE Std 829-1983 is shown in following figure.

1. **Test Plan Identifier**

- Each test plan should have a unique identifier so that it can be associated with a specific project and become a part of the project history.
- The project history and all project-related items should be stored in a project database
- Organizational standards should describe the format for the test plan identifier and how to specify versions

Arunai Engineering College

2. Introduction

- Here, the test planner gives an overall description of the project, the software system being developed or maintained, and the software items and/or features to be tested.
- References to related or supporting documents should also be included
- If test plans are developed as multilevel documents, then each plan must reference the next higher level plan for consistency and compatibility reasons.

3. Items to Be Tested

- This is a listing of the entities to be tested and should include names, identifiers, and version/revision numbers for each entity.
- The items listed could include procedures, classes, modules, libraries, subsystems, and systems.
- References to the appropriate documents and the user manual should be included
- The test planner should also include items that will *not* be included in the test effort.

4. Features to Be Tested

- ***Features may be described as distinguishing characteristics of a software component or system.***
- Features that will *not* be tested should be identified and reasons for their exclusion from test should be included.
- References to test design specifications for each feature and each combination of features are identified

5. Approach

- Provides broad coverage of the issues to be addressed when testing the target software.
- Testing activities are described.
- Tools and techniques necessary for the tests should be included.
- Expectations for test completeness and how the degree of completeness will be determined should be described

6. Item Pass/Fail Criteria

- Given a test item and a test case, the tester must have a set of criteria to decide on whether the test has been passed or failed upon execution.
- A failure occurs when the actual output produced by the software does not agree with what was expected, under the conditions specified by the test.
- Scales are used to rate failures/defects with respect to their impact on the customer/user

7. Suspension and Resumption Criteria

- In the simplest of cases testing is suspended at the end of a working day and resumed the following morning.
- The test plan should also specify conditions to suspend testing based on the effects or criticality level of the failures/defects observed.
- Conditions for resuming the test after there has been a suspension should also be specified.

Arunai Engineering College

8. Test Deliverables

- Test cases describe the actual test inputs and expected outputs.
- Deliverables may also include other documents that result from testing such as test logs, test transmittal reports, test incident reports, and a test summary report.
- Another test deliverable is the test harness that is supplementary code that is written specifically to support the test efforts
- Support code, like, testing tools that will be developed especially for this project, should also be described

9. Testing Tasks

- Identify all testing-related tasks and their dependencies using a Work Breakdown Structure (WBS)
- ***A Work Breakdown Structure is a hierarchical or treelike representation of all the tasks that are required to complete a project.***
- High-level tasks sit at the top of the hierarchical task tree.
- Leaves are detailed tasks sometimes called work packages that can be done by 1–2 people in a short time period, typically 3–5 days.

10. The Testing Environment

- Here the test planner describes the software and hardware needs for the testing effort (Eg.) emulators, telecommunication equipment, etc.
- The planner must also indicate any laboratory space containing the equipment that needs to be reserved.
- The planner also needs to specify any special software needs such as coverage tools, databases, and test data generators.

11. Responsibilities

- The staff who will be responsible for test-related tasks should be identified.
- This includes personnel like, developers, testers, software quality assurance staff, systems analysts, and customers/users, who will be:
  - ✓ developing test design specifications, and test cases;
  - ✓ executing the tests and recording results;
  - ✓ checking results;
  - ✓ interacting with developers;
  - ✓ developing the test harnesses;
  - ✓ interacting with the users/customers.

12. Staffing and Training Needs

- The test planner should describe the staff and the skill levels needed to carry out test-related responsibilities
- Any special training required to perform a task should be noted.

13. Scheduling

- Task durations should be established and recorded with the aid of a task networking tool.
- Test milestones should be established, recorded, and scheduled.

Arunai Engineering College

- Schedules for use of staff, tools, equipment, and laboratory space should also be specified.

14. Risks and Contingencies

- Every testing effort has risks associated with it.
- Testing software with a high degree of criticality, complexity, or a tight delivery deadline all impose risks that may have negative impacts on project goals.
- These risks should be: (i) identified, (ii) evaluated in terms of their probability of occurrence, (iii) prioritized, and (iv) contingency plans should be developed that can be activated if the risk occurs.

15. Testing Costs

- The project manager in consultation with developers and testers estimates testing costs.
- If the test plan is an independent document prepared by the testing group and has a cost component, the test planner will need tools and techniques to help estimate test costs.
- Test costs that should included in the plan are:
    - ✓ costs of planning and designing the tests;
    - ✓ costs of acquiring the hardware and software necessary for the tests;
    - ✓ costs to support the test environment;
    - ✓ costs of executing the tests;
    - ✓ costs of recording and analyzing test results;
    - ✓ tear-down costs to restore the environment.

16. Approvals

- The test plan(s) for a project should be reviewed by those designated by the organization.
- All parties who review the plan and approve it should sign the document.

Test Plan Components
1. Test plan identifier
2. Introduction
3. Items to be tested
4. Features to be tested
5. Approach
6. Pass/fail criteria
7. Suspension and resumption criteria
8. Test deliverables
9. Testing Tasks
10. Test environment
11. Responsibilities
12. Staffing and training needs
13. Scheduling
14. Risks and contingencies
15. Testing costs
16. Approvals

Arunai Engineering College

TEST PLAN ATTACHMENTS

Test Design Specifications

- The IEEE standard for software test documentation describes a test design specification as a test deliverable that specifies the requirements of the test approach.
- The test design specification also has links to the associated test cases and test procedures needed to test the features, and also describes in detail pass/fail criteria for the features.
- To develop test design specifications many documents such as the requirements, design documents, and user manual are useful.
- A test design specification should have the following components according to the IEEE standard.

Test Design Specification Identifier → Give each test design specification a unique identifier and a reference to its associated test plan.

Features to Be Tested → Test items, features, and combination of features covered by this test design specification are listed.

Approach Refinements → In the test plan a general description of the approach to be used to test each item was described.
The test planner also describes how test results will be analyzed.
The relationships among the associated test cases are discussed.

Test Case Identification → Each test design specification is associated with a set of test cases and a set of set procedures.
The test cases contain input/output information
Test procedures contain the steps necessary to execute the tests.

Pass/Fail Criteria → The specific criteria to be used for determining whether the item has passed/failed a test.

Test Case Specifications

- This series of documents attached to the test plan defines the test cases required to execute the test items named in the associated test design specification.
- Each test case must be specified correctly so that time is not wasted in analyzing the results of an erroneous test.

Test Case Specification Identifier → Each test case specification should be assigned a unique identifier.

Test Items → This component names the test items and features to be tested by this test case specification.

Input Specifications → This component of the test design specification contains the actual inputs needed to execute the test.

Arunai Engineering College

Output Specifications → All outputs expected from the test should be identified. If an output is to be a specific value or a specific feature it also should be stated. The output specifications are necessary to determine whether the item has passed/failed the test.

Special Environmental Needs → Any specific hardware and specific hardware configurations needed to execute this test case should be identified. Special software required executing the test such as compilers, simulators, and test coverage tools should be described.

Special Procedural Requirements → Describe any special conditions or constraints that apply to the test procedures associated with this test.

Intercase Dependencies → The test planner should describe any relationships between this test case and others, and the nature of the relationship.

<u>Test Procedure Specifications</u>

A procedure in general is a sequence of steps required to carry out a specific task.

- The planner specifies the steps required to execute a set of test cases.
- It specifies the steps necessary to analyze a software item in order to evaluate a set of features.

Test Procedure Specification Identifier → Each test procedure specification should be assigned a unique identifier.

Purpose → Describe the purpose of this test procedure and reference any test cases it executes.

Specific Requirements → List any special requirements for this procedure, like software, hardware, and special training.

Procedure Steps → Here the actual steps including methods, documents for recording (logging) results, and recording incidents are described.

- Steps include:
  - Setup : to prepare for execution of the procedure;
  - Start : to begin execution of the procedure;
  - Proceed : to continue the execution of the procedure;
  - Measure : to describe how test measurements related to outputs will be made;
  - Shut down : to describe actions needed to suspend the test when unexpected events occur;
  - Restart : to describe restart points and actions needed to restart the procedure from these points;
  - Stop : to describe actions needed to bring the procedure to an orderly halt;

Arunai Engineering College

LOCATING TEST ITEMS

The Test Item Transmittal Report

Suppose a tester is ready to run tests on an item on the date described in the test plan. She needs to be able to locate the item and have knowledge of its current status. This is the ***function of the Test Item Transmittal Report***.

This document is not a component of the test plan, but is necessary to locate and track the items that are submitted for test. Each Test Item Transmittal Report has a unique identifier. It should contain the following information for each item that is tracked.

(i) version/revision number of the item;
(ii) location of the item;
(iii) persons responsible for the item (e.g., the developer);
(iv) references to item documentation and the test plan it is related to;
(v) status of the item;
(vi) approvals — space for signatures of staff who approve the transmittal.

TEST MANAGEMENT

Test managements includes aspects that should be taken care of in planning a project. These aspects are proactive measures that can have an across-the-board influence in all testing projects.

Choice of Standards

Standards are of 2 types, external and internal standards. External standards are standards that a product should comply with, are externally visible, and are usually stipulated by external consortia. Internal standards are standards formulated by a testing organization to bring in consistency and predictability. Some of the internal standards include,

(i) Naming and storage conventions for test artifacts

- Every test artifact should be named appropriately and meaningfully.
- Stipulates the conventions for directory structure for tests.

(ii) Document standards

- For manual testing, documentation standards correspond to specifying the user and system responses at the right level of detail that is consisten with the skill level of the tester.

(iii) Test coding standards

- Test coding standards go one level deeper into the tests and enforce standards on how the tests themselves are written.

(iv) Test reporting standards

- The stakeholders must get a consistent and timely view of the progress of tests.
- It provides guidelines on the level of detail that should be present in the test reports.

Arunai Engineering College

Test Infrastructure Management

Testing requires a robust infrastructure to be planned upfront. This infrastructure is made up of 3 essential elements.

- A test case database (TCDB) → captures all the relevant information aboult the test cases in an organization.
- A defect repository → captures all the relevant details of defects reported for a product. It is an important vehicle of communication that influences the work flow ithin a software organization.
- Configuration management repository and tool → keeps trace of change control and version control of all the files / entities that make up a software product.

Test People Management

- People management is an integral part of any project management.
- A person relies only on his or her own skills to accomplish an assigned activity
- It requires the ability to taught (unlike technical skills).
- Success of a testing organization depends vitally on judicious people managemtn skills.
- The common goals and the spirit of teamwork have to be internalized by all the stakeholders.

Integrating with Product Release

- The success of a product depends on the effectiveness of integration of the development and testing activities.
- Project planning for the entire product should be done in a holistic way.
- Some of the points to be decided for this planning are as follows.
  - Sync points between development and testing as to when different types of testing can commence. (Eg. When integration testing could start? When system testing could start?)
  - Services level agreements between development and testing as to how long it would take for the testing team to complete the testing.
  - Consistent definitions of the various properties and severities of the defects.
  - Communication mechanism to the documentation group to ensure that the documentation is kept in sync with the product.

TEST PROCESS

- **Putting Together and Baselining a Test Plan**
  - An organization develops a template that is to be used across the board and each testing project puts together a test plan based on it.
  - A change, if any, is made only after careful deliberations (proper approval).

Arunai Engineering College

- The test plan is reviewed by a designated set of competent people of the organization.
- It is approved by a competent authority, an independent of project manager directly responsible for testing.
- The test plan is base lined into the configuration management repository.
- Now, the base lined test plan becomes the bases for running the testing project.

- **Test case Specification**
  - **Test case specification is designed by the testing team based on test plan.**
  - **It becomes the bases for preparing individual test cases.**
  - A test case specification should clearly identify,
    - The purpose of test
    - Items being testing along with their version or release number
    - Environment that needs to be set up for running the test case.
    - Input data to be used for the test case.
    - Steps to be followed to execute the test.
    - The expected results that are considered to be "correct results".
    - A step to compare the actual results produced with the expected results.
- Update of Traceability Matrix
  - The traceability matrix is a tool to validate that every requirement is tested.
  - It is created during the requirements gathering phase itself by filling up the unique identifier for each requirement.
  - On completion, the row corresponding to the requirement which is being tested by the test case is updated with the test case specification identifier.
- Identifying Possible Candidates for Automation.
  - Before writing test cases, decision should be taken as to which tests are to be automated and which should run manually.
  - Some criteria that will be used in deciding for automate include,
    - Repetitive nature of the test
    - Effort involved in automation
    - Amount of manual intervention required for the test
    - Cost of automation tool
- Developing and Baselining Test Cases
  - Test case development entails translating the test specifications to a form from which the tests can be executed.
    - For automation, it requires writing test scripts in the automation language.
    - For manual, it maps to writing detailed step-by-step instructions for executing the tests and validating the results.
    - Test case should have change history documents, which specifies,
      - What was the change
      - Why the change was necessitated
      - Who made the change
      - When was the change made
      - Brief description of how the change has been implemented
      - Other files affected by the change

Arunai Engineering College

- Collecting and Analyzing Metrics
  - Information about test execution gets collected in test logs and other files.
- Preparing Test Summary Report
  - On test cycle completion, a test summary report is produced.
  - It gives insights to the senior management about the fitness of the product for release.
- Recommending Product Release Criteria
  - Defect identification is an evidence of what defects exist in the product, their severity and impact.
    - What defects the product has
    - What is the impact / severity of each of the defects
    - What would be the risks of releasing the product with the existing defects?

REPORTING TEST RESULTS

The test plan and its attachments are test-related documents that are prepared *prior* to test execution. There are additional documents related to testing that are prepared during and after execution of the tests. The *IEEE Standard for Software Test Documentation* describes the following documents.

Test Log

The test log, **a diary of events during tests**, should be prepared by the person executing the tests. In the experimental world of engineers and scientists detailed logs are kept when carrying out experimental work.

The test log is invaluable for use in defect repair. It gives the developer a snapshot of the events associated with a failure. The combination of test log and test incident documents helps to prevent incorrect decisions based on incomplete or erroneous test results that often lead to repeated, but ineffective, test-patch-test cycles.

The test log is valuable for (i) regression testing that takes place in the development of future releases of a software product, and (ii) circumstances where code from a reuse library is to be reused.

The test log can have many formats. An organization can design its own format or adopt IEEE recommendations. The IEEE Standard for Software Test Documentation has the following sections:

1. *Test Log Identifier:* Each test log should have a unique identifier.

2. *Description:* Tester should identify the items being tested, their version/revision number, and their associated Test Item/Transmittal Report. The environment in which the test is conducted should be described including hardware and operating system details.

Arunai Engineering College

3. *Activity and Event Entries:* The tester should provide dates and names of test log authors for each event and activity. This section should also contain:

- *Execution description:* Provide a test procedure identifier and also the names and functions of personnel involved in the test.
- *Procedure results:* For each execution, record the results and the location of the output.
- *Incident report identifiers:* Record the identifiers of incident reports generated while the test is being executed.

Test Incident Report

The tester should record in a test incident report (sometimes called a **problem report**) any event that occurs during the execution of the tests that is ***unexpected, unexplainable, and that requires a follow-up investigation***.

The *IEEE Standard for Software Test Documentation* recommends the following sections in the report:

1. *Test Incident Report identifier:* to uniquely identify this report.
2. *Summary:* to identify the test items involved, the test procedures, test cases, and test log associated with this report.
3. *Incident description:* to describe time and date, testers, observers, environment, inputs, expected outputs, actual outputs, procedure step etc.
4. *Impact:* what impact will this incident have on the testing effort, the test plans, the test procedures, and the test cases? (severity rating)

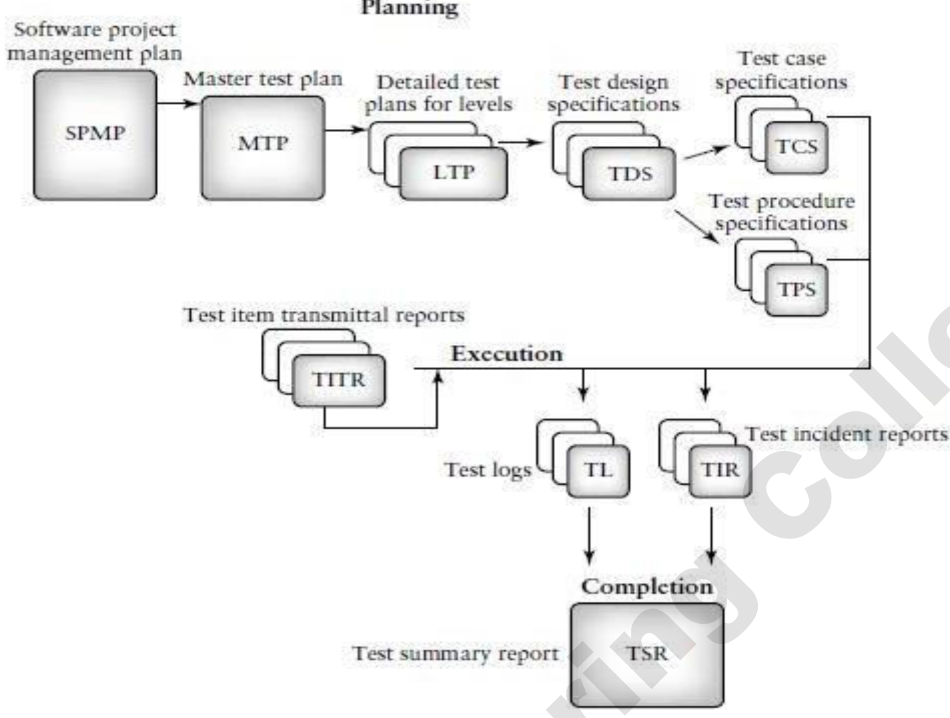
Test Summary Report

This report is prepared when testing is complete, a summary of the results of the testing efforts. When a project postmortem is conducted, the Test Summary Report can help managers, testers, developers, and SQA staff to evaluate the effectiveness of the testing efforts. The IEEE test documentation standard describes the following sections for the Test Summary Report:

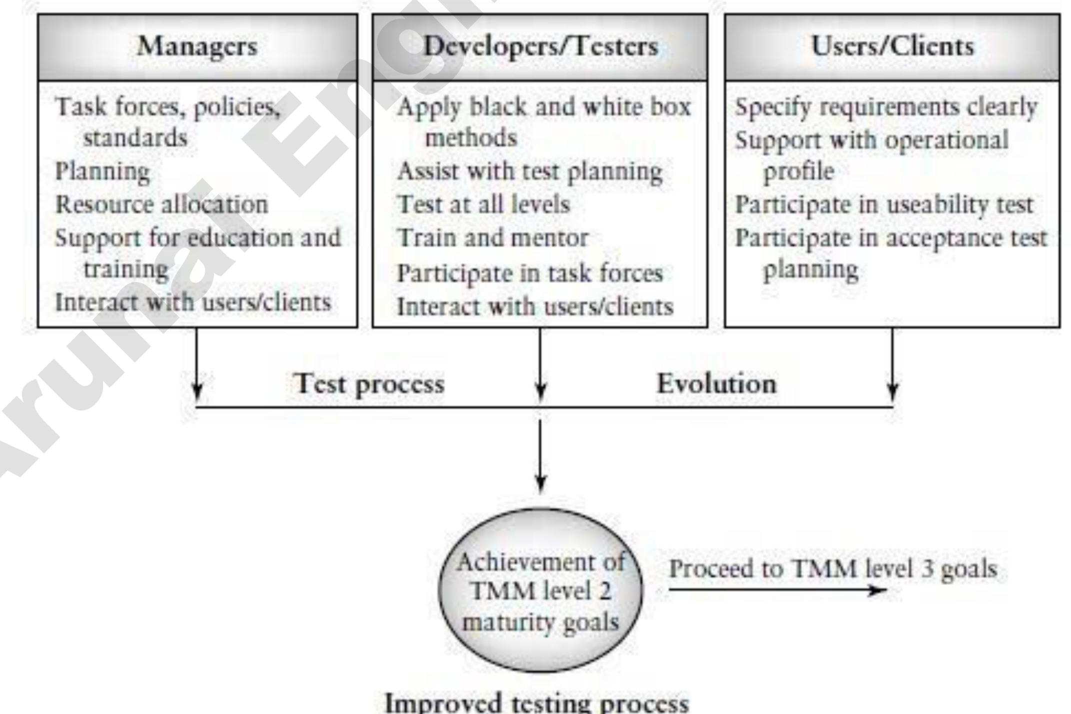
1. *Test Summary Report identifier:* to uniquely identify this report.
2. *Variances:* variances of the test items from their original design. Deviations and reasons for the deviation from the test plan, test procedures, and test designs are discussed.
3. *Comprehensiveness assessment:* the document author discusses the comprehensiveness of the test effort as compared to test objectives and test completeness criteria as described in the test plan.
4. *Summary of results:* the document author summarizes the testing results. Resolved and unresolved incidents should be described.
5. *Evaluation:* author evaluates each test item based on test results. Did it pass/fail the tests? If it failed, what was the level of severity of the failure?
6. *Summary of activities:* all testing activities and events are summarized.
7. *Approvals:* the names of all persons who are needed to approve this document are listed with space for signatures and dates.

Arunai Engineering College

The following figure shows the relationships between all the test-related documents.



THE ROLE OF THE THREE GROUPS IN TESTING PLANNING AND POLICY DEVELOPMENT



Arunai Engineering College

The **manager's view** involves commitment and support for those activities and tasks related to improving testing process quality.
The **developer/tester's view** encompasses the technical activities and tasks that when applied, constitute best testing practices.
The **user/client view** is defined as a cooperating or supporting view.

Critical group participation is summarized in following figure.

INTRODUCING THE TEST SPECIALIST

The organization tests its software at several levels (unit, integration, system, etc.) Moving up to next level requires further investment of organizational resources in the testing process. One of the maturity goals at this level calls for the "Establishment of a test organization." It implies a commitment to better testing and higher-quality software. This commitment requires that testing specialists be hired, space be given to house the testing group, resources be allocated to the group, and career paths for testers be established.

Although there are many costs to establishing a testing group, there are also many benefits. By supporting a test group an organization acquires *leadership* in areas that relate to testing and quality issues. For example, there will be staff with the necessary skills and motivation to be responsible for:

- maintenance and application of test policies;
- development and application of test-related standards;
- participating in requirements, design, and code reviews;
- test planning;
- test design;
- test execution;
- test measurement;
- test monitoring (tasks, schedules, and costs);
- defect tracking, and maintaining the defect repository;
- acquisition of test tools and equipment;
- identifying and applying new testing techniques, tools, and methodologies;
- mentoring and training of new test personnel;
- test reporting.

The staff members of such a group are called test specialists or test engineers. Their primary responsibly is to ensure that testing is effective and productive, and that quality issues are addressed.

Testers are not developers, or analysts, although background in these areas is very helpful and necessary. Testers don't repair code. However, they add value to a software product in terms of higher quality and customer satisfaction. They are not destructive; they are constructive.

Test specialists need to be educated and trained in testing and quality issues.

Arunai Engineering College

SKILLS NEEDED BY A TEST SPECIALIST

Given the nature of assigned to the tester, many managerial and personal skills are necessary for success in the area of work. On the *personal* and *managerial* level a test specialist must have various skill and all of these skills are summarized as follows.

Test specialist skills

Personal and Managerial Skills
Organizational, and planning skills
Track and pay attention to detail
Determination to discover and solve problems
Work with others, resolve conflicts
Mentor and train others
Work with users/clients
Written/oral communication skills
Think creatively

Technical Skills
General software engineering principles and practices
Understanding of testing principles and practices
Understanding of basic testing strategies, and methods
Ability to plan, design, and execute test cases
Knowledge of process issues
Knowledge of networks, databases, and operating systems
Knowledge of configuration management
Knowledge of test-related documents
Ability to define, collect, and analyze test measurements
Ability, training, and motivation to work with testing tools
Knowledge of quality issues

BUILDING A TESTING GROUP

Organizing, staffing, and directing were major activities required to manage a project and a process. Hiring staff for the testing group, organizing the testing staff members into teams, motivating the team members, and integrating the team into the overall organizational structure are organizing, staffing, and directing activities your organization will need to perform to build a managed testing process.

Establishing a specialized testing group is a major decision for an organization. The steps in the process are summarized in following figure.

The following gives a brief description of the duties for each tester that is common to most organizations.

Arunai Engineering College

The Test Manager

In organizations with a testing function, the test manager (or test director) is the central person concerned with all aspects of testing and quality issues. The test manager is usually responsible for,

- test policy making,
- customer interaction,
- test planning,
- test documentation,
- controlling and monitoring of tests,
- training,
- test tool acquisition,
- participation in inspections and walkthroughs,
- reviewing test work,
- the test repository,
- and staffing issues such as hiring, firing, and evaluation of the test team members.

Arunai Engineering College

The Test Lead

The test lead assists the test manager and works with a team of test engineers on individual projects. He or she may be responsible for duties such as,

(v) test planning, (ii) staff supervision, and (iii) status reporting.

The test lead also participates in test design, test execution and reporting, technical reviews, customer interaction, and tool training.

The Test Engineer

The test engineers design, develop, and execute tests, develop test harnesses, and set up test laboratories and environments. They also give input to test planning and support maintenance of the test and defect repositories.

The Junior Test Engineer

The junior test engineers are usually new hires. They gain experience by participating in test design, test execution, and test harness development. They may also be asked to review user manuals and user help facilities defect and maintain the test and defect repositories.

Arunai Engineering College

ARUNAI ENGINEERING COLLEGE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
IT8076 -SOFTWARE TESTING (2017 Regulation)
UNIT-V

UNIT V TEST AUTOMATION

Software test automation – skills needed for automation – scope of automation – design and architecture for automation – requirements for a test tool – challenges in automation – Test metrics and measurements – project, progress and productivity metrics.

Software Test Automation

WHAT IS TEST AUTOMATION?

Developing software to test the software is called test automation.

Test automation can help address several problems.

- **Automation save time as software can execute test cases faster than human do** . The time thus saved can be used effectively for test engineers to

1. develop additional test cases to achieve better coverage;
2. perform some esoteric or specialized tests like ad hoc testing; or
3. Perform some extra manual testing.

   The time saved in automation can also be utilized to develop additional test cases, thereby improving the coverage of testing.

- **Test automation can free the test engineers from mundane tasks and make them focus on more creative tasks. -**E.g- Ad hoc testing requires intuition and creativity to test the product for those perspectives that may have been missed out by planned test cases. If there are too many planned test cases that need to be run manually and adequate automation does not exist, then the test team may spend most of its time in test execution.
  Automating the more mundane tasks gives some time to the test engineers for creativity and challenging tasks.
- **Automated tests can be more reliable** -when an engineer executes a particular test case many times manually, there is a chance for human error. As with all machine-oriented activities, automation can be expected to produce more reliable results every time, and eliminates the factors of boredom and fatigue.
- **Automation helps in immediate testing** -automation reduces the time gap between development and testing as scripts can be executed as soon as the product build is ready. Automated testing need not wait for the availability of test engineers.
- **Automation can protect an organization against attrition of test engineers** Automation can also be used as a knowledge transfer tool to train test engineers on the product as it has a repository of different tests for the product.
- **Test automation opens up opportunities for better utilization of global resources** Manual testing requires the presence of test engineers, but automated tests can be run round the clock, twenty- four hours a day and seven days a week.

Arunai Engineering College

This will also enable teams in different parts of the words, in different time zones, to monitor and control the tests, thus providing round the- clock coverage.

- **Certain types of testing cannot be executed without automation-**For example, if we want to study the behavior of a system with thousands of users logged in, there is now way one can perform these tests without using automated tolls.
- **Automation means end-to-end, not test execution alone -**Automation should consider all activities such as picking up the right product build, choosing the right configuration, performing installation, running the tests, generating the right test data, analyzing the results, and filling the defects in the defect repository. When talking about automation, this large picture should always be kept in mind.

TERMS USED IN AUTOMATION

A test case is a set of sequential steps to execute a test operating on a set of predefined inputs to produce certain expected outputs. There are two types of test cases – automated and manual. **A manual test case** is executed manually while an **automated test** case is executed using automation.

As we have seen earlier , testing involves several phases and several types of testing. Some test cases are repeated several times during a product release, because the product is built several times. Table describes some test cases for the log in example, on how the login can be tested for different types of testing.

S.No	Test Cases for Testing	Belongs to What type of testing
1.	Check whether login works	Functionality
2.	Repeat Login operation in a loop for 48 hours	Reliability
3.	Perform Login from 10000 clients	Load /Stress Testing
4.	Measure time taken for Login operations in different conditions	Performance
5.	Run log in operation from a machine running Japanese language	Internationalization

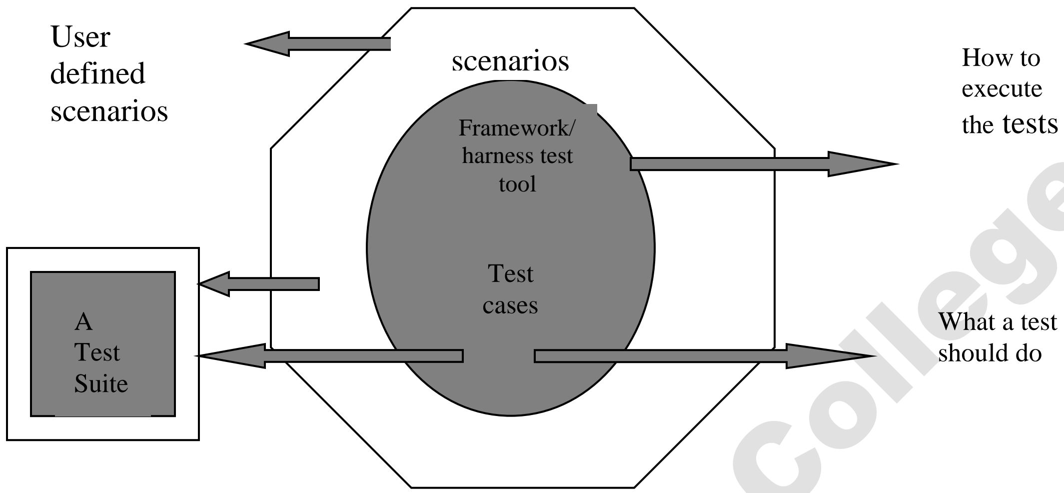
From the above table , it is observed that there are 2 important dimensions

1) What operations have to be tested
2) How the operations have to be tested → scenarios

The how portion is called → scenarios.
What an operation has to do → product specific feature
How they are to be run → framework specific requirement
They are the generic requirements for all products that are being tested in an organization.

Arunai Engineering College

When a set of test cases is combined and associated with a set of scenarios, they are called "**test suite**". A test suite is nothing but a set of test cases that are automated and scenarios that are associated with the test cases.



Framework for test automation

SKILLS NEEDED FOR AUTOMATION

There are different "Generations of Automation". The skills required for automation depends on what generation automation the company is in or desires to be in the near future.

The automation of testing is broadly classified into three generations.

a. **First generation – Record and playback**
   - Record and playback avoids the repetitive nature of executing tests. Almost all the test tools available in the market have the record and playback feature.
   - A test engineer records the sequence of actions by keyboard characters or mouse clicks and those recorded scripts are played back later, in the same order as they were recorded. But this generation of tool has several disadvantages.
   - The scripts may contain hard-coded values, thereby making it difficult to perform general types of tests.
   - When the application changes, all the scripts have to be re-recorded, thereby increasing the test maintenance costs.

b. **Second generation-Data-driven**
   - This method helps in developing test scripts that generates the set of input conditions and corresponding expected output.
   - This enables the tests to be repeated for different input and output conditions. The approach takes as much time and effort as the product.

Arunai Engineering College

c. **Third generation-Action-driven**
- This technique enables a layman to create automated tests. There are no input and expected output conditions required for running the tests.
- All actions that appear on the application are automatically tested, based on a generic set of controls defined for automation.
- The set of actions are represented as objects and those objects are reused. The user needs to specify only the operations and everything else that is needed for those actions are automatically generated.
- Hence, automation in the third generation involves two major aspects- "test case automation" and "framework design".

Classification of skills for automation

Automation-first generation	**Automation- second generation**	**Automation-third generation**	
Skills for test case automation	**Skills for test case automation**	**Skills for test case automation**	**Skills for framework**
Scripting languages	Scripting languages	Scripting languages	Programming languages
Record- playback tools usage	Programming languages	Programming languages	Design and architecture skills for framework creation
	Knowledge of data generation techniques	Design and architecture of the product under test	Generic test requirements for multiple products
	Usage of the product under test	Usage of the framework	

<u>**SCOPE OF AUTOMATION**</u>

1. Identifying the Types of Testing Amenable to Automation

Certain types of tests automatically lend themselves to automation

a. **Stress, reliability, scalability, and performance testing** these types of testing require the test cases to be run form a large number of different machines for an extended period of time, such as 24 hours, 48 hours, and so on. Test cases belonging to these testing types become the first candidates for automation.

b. **Regression tests** Regression tests are repetitive in nature. These test cases are executed multiple times during the product development phases.

c. **Functional tests** These kinds of tests may require a complex set up and thus require specialized skill, which may not be available on an ongoing basis. Automating these once, using the expert skill sets, can enable using less-skilled people to run these tests on an ongoing basis.

2. Automating Areas Less Prone To Change

Automation should consider those areas where requirements go through lesser or no changes. Normally change in requirements cause scenarios and new features to be impacted, not the basic functionality of the product.

Arunai Engineering College

3. **Automate Tests That Pertain to Standards**

One of the tests that product may have to undergo is compliance to standards. For example, a product providing a JDBC interface should satisfy the standard JDBC tests.

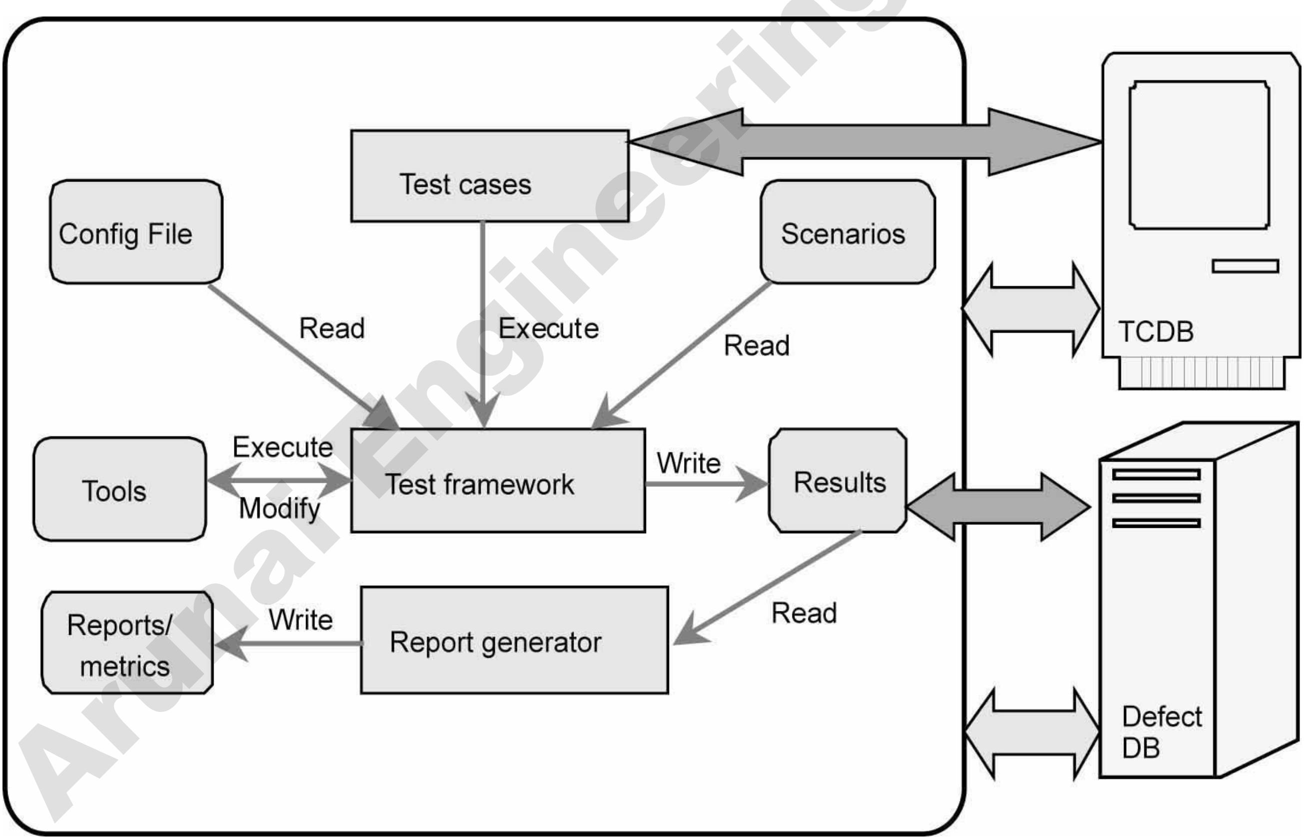
Automating for standards provides a dual advantage. Test suites developed for standards are not only used for product testing but can also be sold as test tools for the market.

4. **Management Aspects in Automation**

Prior to starting automation, adequate effort has to be spent to obtain management commitment. It involves significant effort to develop and maintain automated tools; obtaining management commitment is an important activity. Return on investment is another aspect to be considered seriously.

DESIGN AND ARCHITECTURE FOR AUTOMATION

Design and architecture is an important aspect of automation. As in product development, the design has to represent all requirements in modules and in the interactions between modules and in the interactions between modules.



Integration Testing, both internal interfaces and external interfaces have to be captured by design and architecture. In this figure the thin arrows represent the internal interfaces and the direction of flow and thick arrows show the external interfaces. All the

Arunai Engineering College

modules, their purpose, and interactions between them are described in the subsequent sections.
Architecture for test automation involves two major heads: a test infrastructure that covers a test case database and a defect database or defect repository. Using this infrastructure, the test framework provides a backbone that ties the selection and execution of test cases.

1. External Modules

- There are two modules that are external modules to automation-TCDB and defect DB. All the test cases, the steps to execute them, and the history of their execution are stored in the TCDB.
- The test cases in TCDB can be manual or automated. The interface shown by thick arrows represents the interaction between TCDB and the automation framework only for automated test cases.
- Defect DB or defect database or defect repository contains details of all the defects that are found in various products that are tested in a particular organization. It contains defects and all the related information test engineers submit the defects for manual test cases.
- For automated test cases, the framework can automatically submit the defects to the defect DB during execution.

2. Scenario and Configuration File Modules.

Scenarios are nothing but information on "how to execute a particular test case".
A configuration file contains a set of variables that are used in automation. A configuration file is important for running the test cases for various execution conditions and for running the tests for various input and output conditions and states.

3. Test Cases and Test Framework Modules

Test case is an object for execution for other modules in the architecture and does not represent any interaction by itself.

A test framework is a module that combines "what to execute" and "how they have to be executed". It picks up the specific test cases that are automated from TCDB and picks up the scenarios and executes them.

The test framework is considered the core of automation design. It subjects the test cases to different scenarios. The test framework contains the main logic for interacting , initiating, and controlling all modules.

A test framework can be developed by the organization internally or can be bought from the vendor.

4. Tools and Result Modules

- When a test framework performs its operations, there are a set of tools that may be required. For example, when test cases are stored as source code files in TCDB, they need to be extracted and compiled by build tools. In order to run the compiled code, certain runtime tools and utilities may be required.
- For eg , IP Packet Simulators. The result that comes out of the tests run by the test framework should not overwrite the results from the previous test runs. The history of all the previous tests run should be recorded and kept as archives.

Arunai Engineering College

5. Report Generator and Reports / Metrics Modules

- Once the results of a test run are available, the next step is to prepare the test reports and metrics. Preparing reports is a complex and time- consuming effort and hence it should be part of the automation design.
- There should be customized reports such as an executive report, which gives very high level status; technical reports, which give a moderate level of detail of the test run; and detailed or debug reports which are generated for developers to debug the failed test cases and the product.
- The module that takes the necessary inputs and prepares a formatted report is called a report generator. Once the results are available, the report generator can generate metrics.

GENERIC REQUIREMENTS FOR TEST TOOL/FRAMEWORK

Requirement 1: No hard coding in the test suite

- The variables for the test suite are called configuration variables. The file in which all variable names and their associated values are kept is called configuration file.
- The variables belonging to the test tool and the test suite need to be separated so that the user of the test suite need not worry about test tool variables.
- Changing test tool variables, without knowing their purpose, may impact the results of the tests.
- Providing inline comment for each of the variables will make the test suite more usable and may avoid improper usage of variables.

Ex: well documented config file

```
#Test Framework Configuration Parameter
TOOL_PATH =/tools
COMMONLIB_PATH =/tools/crm/lib
SUITE_PATH =/tools/crm

#parameter common to all the test cases in the test
VERBOSE_LEVEL =3
MAX_ERRORS=200
USER_PASSWD =hello123

#Test Case1 Parameter
TC1_USR_CREATE =0 # 1=yes 0=no
TC1_USR_PASSWD=hello123
TC1_MAX_USRS =200
```

Requirement 2: Test case/ suite expandability

Points to considered during expansion are

- ❖ Adding a test case should not affect other test cases
- ❖ Adding a test case should not result in retesting the complete test suite
- ❖ Adding a new test suite to the framework should not affect existing test suites

Arunai Engineering College

Requirement 3: Reuse of code for different types of testing, test cases

Points to be considered during Reuse of codes are:

1) The test suite should only do what a test is expected to do. The test framework needs to take care of "how" and

2) The test programs need to be modular to encourage reuse of code.

Requirement 4: Automatic setup and cleanup

When test cases expect a particular setup to run the tests, it will be very difficult to remember each one of them and do the setup accordingly in the manual method. Hence, each test program should have a "setup" program that will create the necessary setup before executing the test cases. The test framework should have the intelligence to find out what test cases are executed and call the appropriate setup program.

A setup for one test case may work negatively for another test case. Hence, it is important not only to create.

Requirement 5: Independent test cases

Each test case should be executed alone; there should be no dependency between test cases such as test case-2 to be executed after test case-1 and so on. This requirement enables the test engineer to select and execute any test case at random without worrying about other dependencies.

Requirement 6: Test case dependency

Making a test case dependent on another makes it necessary for a particular test case to be executed before or after a dependent test case is selected for execution

Requirement 7: Insulating test cases during execution

Insulating test cases from the environment is an important requirement for the framework or test tool. At the time of test case execution, there could be some events or interrupts or signals in the system that may affect the execution.

Requirement 8: Coding standards and directory structure

Coding standards and proper directory structures for a test suite may help the new engineers in understanding the test suite fast and help in maintaining the test suite. Incorporating the coding standards improves portability of the code.

Requirement 9: Selective execution of test cases

A Test Framework contains → many Test Suite
A Test Suite contains → many Test Program
A Test Program contains → many Test Cases

The selection of test cases need not be in any order and any combination should be allowed. Allowing test engineers to select test cases reduces the time. These selections are normally done as part of the scenario file. The selection of test cases can be done dynamically just before running the test cases, by editing the **scenario file**.

Example scenario file	**Meaning**
test-pgm-name 2,4,1,7-10	The test cases 2,4,1,7-10 are selected for execution
Tests-pgm-name	Executes all test cases

Arunai Engineering College

Requirement 10: Random execution of test cases

Test engineer may sometimes need to select a test case randomly from a list of test cases. Giving a set of test cases and expecting the test tool to select the test case is called random execution of test cases. A test engineer selects a set of test cases from a test suite; selecting a random test case from the given list is done by the test tool.

Ex: **scenario file.**

Random test-pgm-name 2,1,5	test tool select one out of test cases 2,1,5 for execution
Random test-pgm-name1 (2,1,5) test-pgm-name2 test-pgm-name3	Test engineer wants one out of test program 1,2,3 to be randomly executed and if pgm-name1 is selected , then one out of test cases 2,1,5 to be randomly executed, if test program 2,3 are selected , then all TC in those 2 program are executed.

Requirement 11: parallel execution of test cases

In a multi-tasking and multi processing operating systems it is possible to make several instances of the tests and make them run in parallel. Parallel execution simulates the behavior of several machines running the same test and hence is very useful for performance and load testing.

Ex: **scenario file.**

Instance, 5 test-pgm-name1 (3)	5 instances of test case 3 in test-pgm-name1 are executed
Instance, 5 test-pgm-name1 (2,1,5) test-pgm-name2 test-pgm-name3	5 instances of test programs are created , within each of the five instances that are created the test program 1,2,3, are executed in sequence .

Requirement 12: Looping the test cases

Reliability testing requires the test cases to be executed in a loop. There are two types of loops that are available.

1) iteration loop - gives the number of iterations of a particular test case to be executed.
2) timed loop - which keeps executing the test cases in a loop till the specified time duration is reached.

Ex: **scenario file.**

Repeat_loop, 50 test-pgm-name1 (3)	test case 3 in test-pgm-name1 is repeated 50 times.
Time_loop, 5 Hours test-pgm-name1 (2,1,5) test-pgm-name2 test-pgm-name3	TC 2,1,5 from test-pgm-name1 and all test cases from the test program2 and 3 are executed in order, in a loop for 5 hours

Arunai Engineering College

Requirement 13: Grouping of test scenarios

The group scenarios allow the selected test cases to be executed in order, random, in a loop all at the same time. The grouping of scenarios allows several tests to be executed in a predetermined combination of scenarios.

Ex: **scenario file.**

Group_scenario1 Parallel, 2 AND repeat,10@scen1 Scen1 test-pgm1 (2,1,5) test-pgm2 test-pgm3	Group scenario was created to execute 2 instances of the individual scenario "scen1" in a loop 10 times

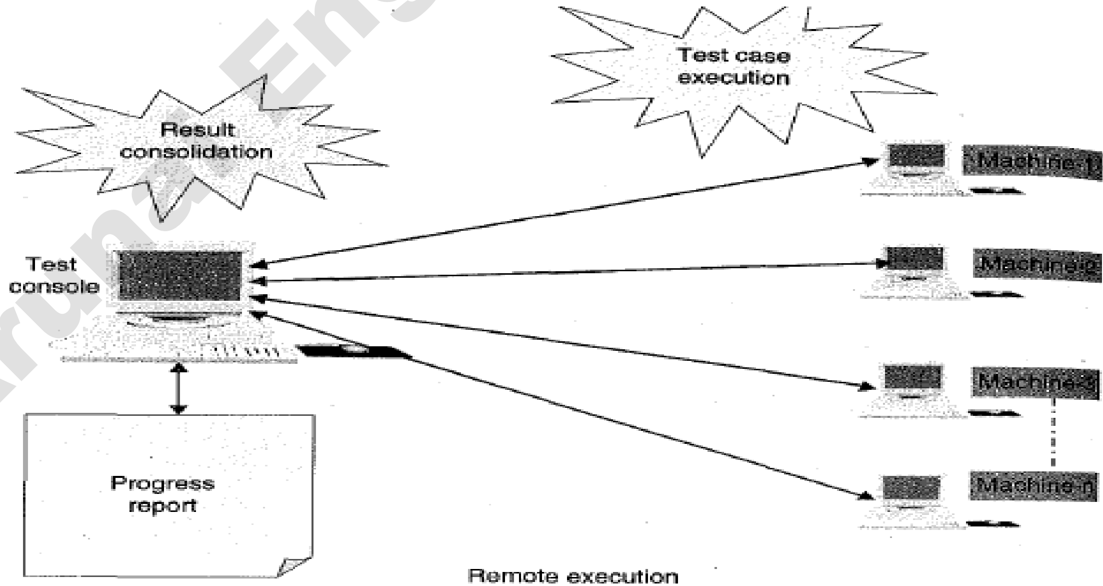
Requirement 14: Test case execution based on previous results

One of the effective practices is to select the test cases that are not executed and test cases that failed in the past and focus more on them. Some of the common scenarios that require test cases to be executed based on the earlier results are

1. Rerun all test cases which were executed previously;
2. Resume the test cases from where they were stopped the previous time;
3. Rerun only failed/not run test cases; and
4. Execute all test cases that were executed previously.

Requirement 15: Remote execution of test cases

The central machine that allocates tests to multiple machines and co-ordinates the execution and result is called test console or test monitor. In the absence of a test console, not only does executing the results from multiple machines become difficult, collecting the results from all those machines also becomes difficult.



Role of test console and multiple execution machine.

Arunai Engineering College

Requirement 16: Automatic archival of test data

The test cases have to be repeated the same way as before, with the same scenarios, same configuration variables and values, and so on. This requires that all the related information for the test cases have to be archived. It includes

1) What configuration variables were used
2) What scenario was used
3) What program were executed and from what path

Requirement 17: Reporting scheme

Every test suite needs to have a reporting scheme from where meaningful reports can be extracted. As we have seen in the design and architecture of framework, the report generator should have the capability to look at the results file and generate various reports. Audit logs are very important to analyze the behavior of a test suite and a product. A reporting scheme should include

1. When the framework, scenario, test suite, test program, and each test case were started/ completed;
2. Result of each test case;
3. Log messages;
4. Category of events and log of events; and
5. Audit reports

Requirement 18: Independent of languages

A framework or test tool should provide a choice of languages and scripts that are popular in the software development area.

- A framework should be independent of programming languages and scripts.
- A framework should provide choice of programming languages, scripts, and their combinations.
- A framework or test suite should not force a language/script.
- A framework or test suite should work with different test programs written using different languages, and scripts.
- The internal scripts and options used by the framework should allow the developers of a test suite to migrate to better framework.

Requirement 19: portability to different platforms

With the advent of platform-independent languages and technologies, there are many products in the market that are supported in multiple OS and language platforms.

- The framework and its interfaces should be supported on various platforms.
- Portability to different platforms is a basic requirement for test tool/ test suite.
- The language/script used in the test suite should be selected carefully so that it runs on different platforms.
- The language/ script written for the test suite should not contain platform-specific calls.

Arunai Engineering College

CHALLENGES IN AUTOMATION

- Test automation presents some very unique challenges. The most important of these challenges is management commitment.
- Automation should not be viewed as a panacea for all problems nor should it be perceived as a quick-fix solution for all the quality problems in a product.
- The main challenge here is because of the heavy front-loading of costs of test automation, management starts to look for an early payback.
- Successful test automation endeavors are characterized by unflinching management commitment, a clear vision of the goals, and the ability to set realistic short-term goals that track progress with respect to the long-term vision.

TEST METRICS AND MEASUREMENTS

Definition

- Metrics are the source of measurement.
- Metrics derive information from raw data with a view to help in decision making.

Ex: No of defects , No of test cases , effort , schedule
Metrics are needed to know test case execution productivity and to estimate test completion date.

Effort : actual time that is spent on a particular activity or a phase Schedule : Elapsed days for a complete set of activities

Steps in a Metrics Program

Identify what to measure
Transform measurements to metrics
Decide operational requirements
Perform metrics analysis
Take actions and follow up
Refine measurements and metrics

Arunai Engineering College

Step1: Metrics program is to decide what measurements are important and collect data accordingly. Ex for Measurements: effort spent on testing , no of defects , no of test cases.

Step2: It deals with defining how to combine data points or measurement to provide meaningful metrics. A particular metric can use one or more measurements

Step3: It involves with operational requirement for measurements. It contains

Who should collect measurements?

Who should receive the analysis etc.

This step helps to decide on the appropriate periodicity for the measurements as well as assign operational responsibility for collecting, recording and reporting the measurements.
Daily basis measurements → no of testcases executed, no of defects found, defects fixed..
Weekly measurements → how may testcases produced 40 defects,

Step4: This step analyzes the metrics to identify both positive area and improvement areas on product quality.

Step5: The final step is to take necessary action and follow up on the action.

Step6: To continue with next iteration of metrics programs, measuring a different set of measurements, leading to more refined metrics addressing different issues.

WHY METRICS IN TESTING?

Knowing only how much testing got completed does not answer the question on when the testing will get completed and when the product is ready for release. To Answer these questions , one need to estimate the following

$$\text{Days needed to complete testing} = \frac{\text{Total test cases yet to be executed}}{\text{Total test case execution productivity}}$$

test case execution productivity → testcases executed per person day

$$\text{Total Days needed for defect fixes} = \frac{\text{(outstanding defects yet to fixed + Defects that can be found in future test cycles)}}{\text{Defect fixing capability}}$$

Days needed for Release = Max(Days needed for testing , days needed for defect fixes)

More accurate estimate with regression testing

Days needed for Release = Max(Days needed for testing ,(days needed for defect fixes + Days needed for regressing outstanding defect fixes))

Metrics are needed to know test case execution productivity and to estimate test completion date.

Annai Engineering College

Metrics in testing help in identifying

- ❖ When to make the release.
- ❖ What to release
- ❖ Whether the product is being released with known quality.

TYPES OF METRICS

Metrics can be classified into different types based on what they measure and what area they focus on. At a very high level, metrics can be classified as product metrics and process metrics.

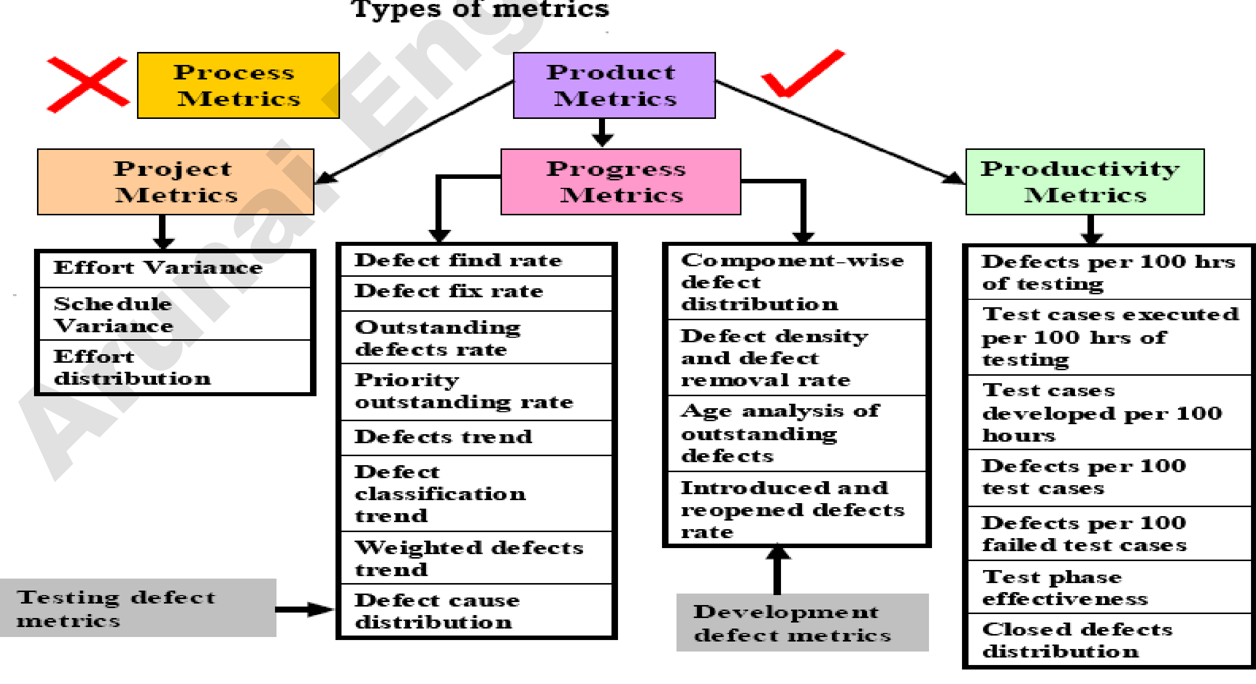
Product metrics can be further classified as:

1. **Project metrics** A set of metrics that indicates how the project is planned and executed.

2. **Progress metrics** A set of metrics that tracks how the different activities of the project are progressing. The activities include both development activities and testing activities. Progress metrics is monitored during testing phases. Progress metrics helps in finding out the status of test activities and they are also good indicators of product quality. Progress metrics, for convenience, is further classified into

   1) test defect metrics and
   2) development defect metrics.

3. **Productivity metrics** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities. These metrics help in planning and tracking testing activities. These metrics help in planning and estimating of testing activities.



Arunai Engineering College

I) PROJECT METRICS

A typical project starts with requirements gathering and ends with product release. All the phases that fall in between these points need to be planned and tracked. In the planning cycle, the scope of the project is finalized. The project scope gets translated to size estimates, which specify the quantum of work to be done. This size estimate gets translated to effort estimate for each of the phases and activities by using the available productivity data available.

base lined effort →The initial effort

revised effort. → As the project progresses and if the scope of the project changes, then the effort estimates are re-evaluated again and this re-evaluated effort estimate is called revised effort.

Two factors
Effort : actual time that is spent on a particular activity or a phase
Schedule : Elapsed days for a complete set of activities

- If the effort is tracked closely & met then schedule can be met.
- If planned effort is equal to actual effort and schedule not met then project is not considered as successful one.
-

The basic measurements are

1. initial baselined effort and schedule
2. The actual effort
3. The revised estimate of effort and schedule

1. Effort Variance (Planned Vs Actual)

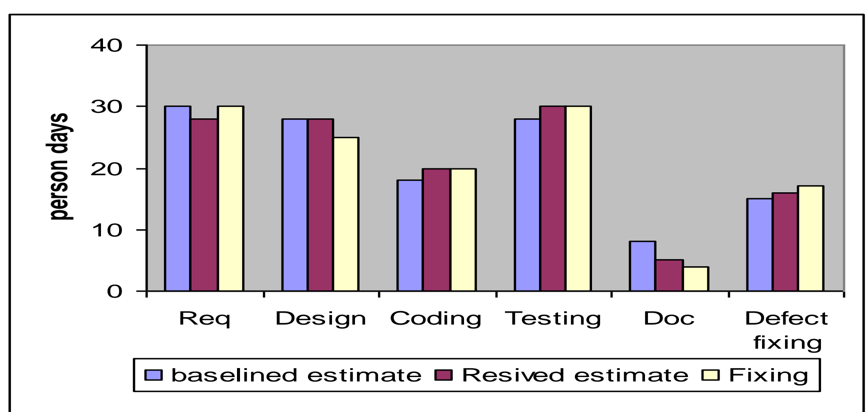
When the baselined effort estimates, revised effort estimates, and actual effort are plotted together for all the phases of SDLC, it provides many insights about the estimation process. As different set of people may get involved in different phases, it is a good idea to plot these effort numbers phase-wise. A sample data for each of the phase is plotted in the chart.

If there is a substantial difference between the baselined and revised effort, it points to incorrect initial estimation. Calculating effort variance for each of the phases provides a quantitative measure of the relative difference between the revised and actual efforts.

Calculating effort variance for each of the phases provides a quantitative measure of the relative difference between the revised and actual efforts.

$$\text{Effort variance \%} = \frac{\text{actual effort} - \text{revised estimate X 100}}{\text{Revised estimate}}$$

Arunai Engineering College



Phase wise effort variation

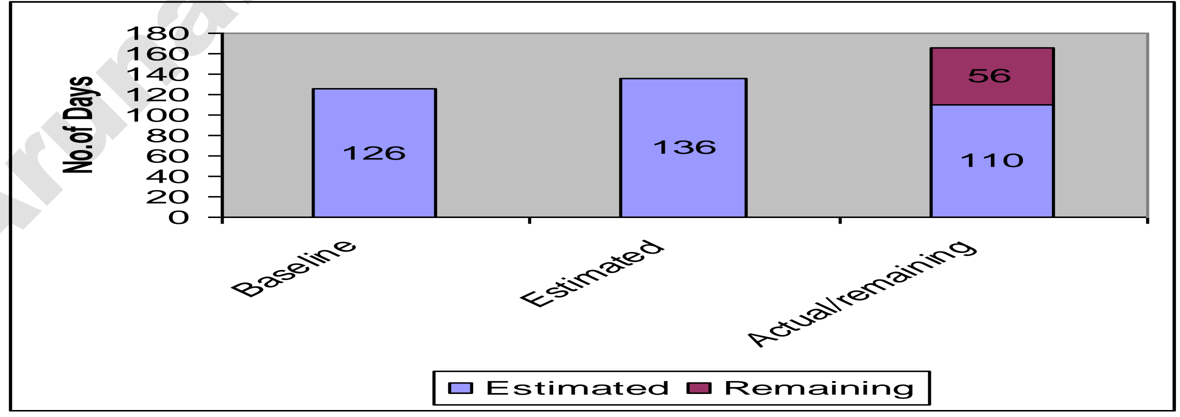
Sample variance percentage by phase.

Effort	**Req**	**Design**	**Coding**	**Testing**	**Doc**	**Defect Fixing**
Variance %	7.1	8.7	5	0	40	15

- All the baseline estimates, revised estimates, and actual effort are plotted together for each of the phases. The variance can be consolidated into as shown in the above table.
- A variance of more than 5% in any of the SDLC phase indicates the scope for improvement in the estimation. The variance is acceptable only for the coding and testing phases.
- The variance can be negative also. A negative variance is an indication of an over estimate.
- The variance is acceptable only for the coding and testing phases.

2. Schedule Variance (Planned vs Actual)

Schedule variance is calculated at the end of every milestone to find out how well the project is doing with respect to the schedule.



To get a real picture on schedule in the middle of project execution, it is important to calculate "remaining days yet to be spent" on the project and plot it along with the "actual

Arunai Engineering College

schedule spent" as in the above chart. "Remaining days yet to be spent" can be calculated by adding up all remaining activities. If the remaining days yet to be spent on project is not calculated and plotted, it does not give any value to the chart in the middle of the project, because the deviation cannot be inferred visually from the chare. The remaining days in the schedule becomes zero when the release is met.

Effort and schedule variance have to be analyzed in totality, not in isolation. This is because while effort is a major driver of the cost, schedule determines how best a product can exploit market opportunities, variance can be classified into negative variance, zero variance, acceptable variance, and unacceptable variance.

Interpretation of range of effort and schedule variation

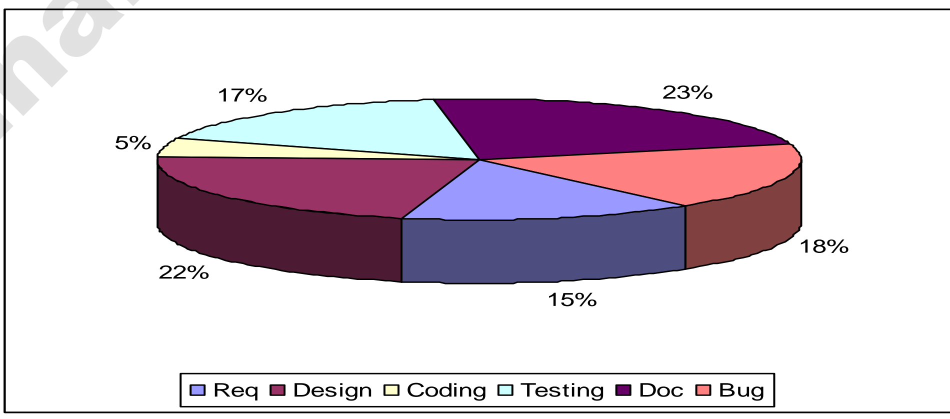
Effort Variance	Schedule Variance	Probable Causes /Result
Zero (or) Acceptable variance (0-5)	Zero variance	A Well executed Project
	Acceptable variance	Need slight improvement
Unacceptable variance (>5)	Zero (or) Acceptable variance	Under estimation , need further analysis
	Unacceptable variance	Under estimation of both effort and schedule
Negative variance	Zero (or) Acceptable variance	Over estimation, need improvement
	Negative variance	Over estimation, need improvement

3. Effort Distribution Across Phases

Adequate and appropriate effort needs to be spent in each of the SDLC phase for a quality product release.

The distribution percentage across the different phases can be estimated at the time of planning and these can be compared with the actual at the time of release for getting a comfort feeling on the release and estimation methods. A sample distribution of effort across phases is given in figure.

Actual Effort Distribution



Effort distribution :

Req > Testing > design > bug fixing > coding > doc

Arunai Engineering College

Mature organizations spend at least 10-15 % of the total effort in requirements and approximately the same effort in the design phase. The effort percentage for testing depends on the type of release and amount of change to the existing code base and functionality. Typically, organizations spend about 20 -50 % of their total effort in testing.

II) PROGRESS METRICS

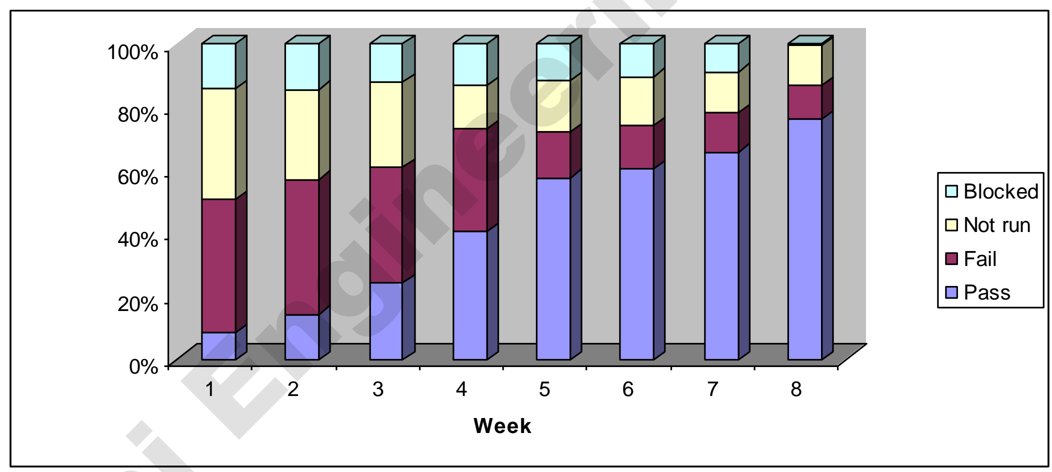
One of the main objectives of testing is to find as many defects as possible before any customer finds them. The number of defects that are found in the product is one of the main indicators of quality.

Defects get detected by the testing team and get fixed by the development team. Defect metrics are further classified in to

1. test defect metrics
2. development defect metrics

The progress chart gives

- pass rate
- fail rate of executed test cases
- pending test cases
- test cases that are waiting for defects to be fixed.



A scenario represented by such a progress chart shows that not only is testing progressing well, but also that the product quality is improving. The chart had shown a trend that as the weeks progress, the "not run" cases are not reducing in number, or "blocked" cases are increasing in number, or "pass" cases are not increasing, then it would clearly point to quality problems in the product that prevent the product from being ready for release.

1. TEST DEFECT METRICS

The next set of metrics helps us understand how the defects that are found can be used to improve testing and product quality.

Some organizations classify effects by assigning a defect priority (for example P1, P2, P3, and so on)Some organizations use defect severity levels (for example, S1, S2, S3, and so

Arunai Engineering College

on).The priority of a defect can change dynamically once assigned. Severity is absolute and does not change often as they reflect the state and quality of the product.

Table -Defect priority and defect severity – sample interpretation.

Defect priority is based on defect fixing and defect severity is based on functionality level.

Priority	**What it means**
1	Fix the defect on highest priority; fix it before the next build
2.	Fix the defect on high priority before next test cycle
3	Fix the defect on moderate priority when time permits, before the release
4	Postpone this defect for next release or live with this defect
Severity	**What it means**
1	The basic product functionality failing or product crashes
2	Unexpected error condition or a functionality not working
3	A minor functionality is failing or behaves differently than expected
4	Cosmetic issue and no impact on the users

This defect classification is based on priority and severity.

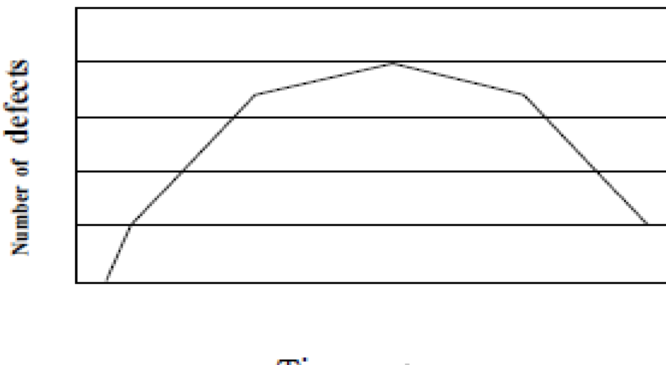
Defect Classification	**What it Means**
Extreme	• Product crashes or unusable • Need to be fixed immediately
Critical	• Basic functionality of the product not working • Needs to be fixed before next test cycle starts
Important	• Extended functionality of the product not working • Does not affect the progress of testing • Fix it before the release
Minor	• Product Behaves differently • No impact on test team or customer • Fix it when time permits
Cosmetic	• Minor Irritant • Need not be fixed for this release

a) Defect Find Rate

The purpose of testing is to find defects early in the test cycle. The idea of testing is to find as many defects as possible early in the cycle. However, this may not be possible for two reasons. First, not all features of a product may become available early; because of scheduling of resources, the features of a product arrive in a particular sequence. Some of the test cases may be blocked because of some show stopper defects.

Once a majority of the modules become available and the defects that are blocking the tests are fixed, the defect arrival rate increases. After a certain period of defect fixing and testing, the arrival of defects tends to slow down and a continuation of that enables product release. This results in a "bell curve" as shown in figure.

Arunai Engineering College



b) Defect fix rate

The purpose of development is to fix defects as soon as they arrive. If the goal of testing is to find defects as early as possible, it is natural to expect that the goal of development should be to fix defects as soon as they arrive. There is a reason why defect fixing rate should be same as defect arrival rate. If more defects are fixed later in the cycle, they may not get tested properly for all possible side-effects.

c) Outstanding defects rate

In a well executed project, the number of outstanding defects is very close to zero all the time during the test cycle. The number of defects outstanding in the product is calculated by subtracting the total defects fixed from the total defects found in the product.

The number of defects outstanding = total defects found in the product - total defects fixed

d) Priority outstanding rate

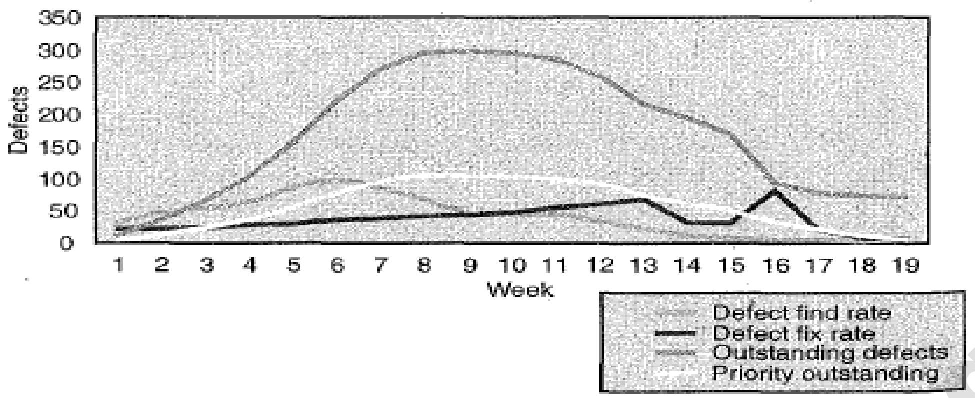
The modification to the outstanding defects rate curve by plotting only the high-priority defects and filtering out the low- priority defects is called priority outstanding defects. This is an important method because closer to product release, the product team would not want to fix the low – priority defects.

Normally only high-priority defects are tracked during the period closer to release. Some high-priority defects may require a change in design or architecture & fixed immediately

e) Defect trend

The effectiveness analysis increases when several perspectives of find rate, fix rate, outstanding, and priority outstanding defects are combined.

Arunai Engineering College



Defect trend

The following observations can be made.

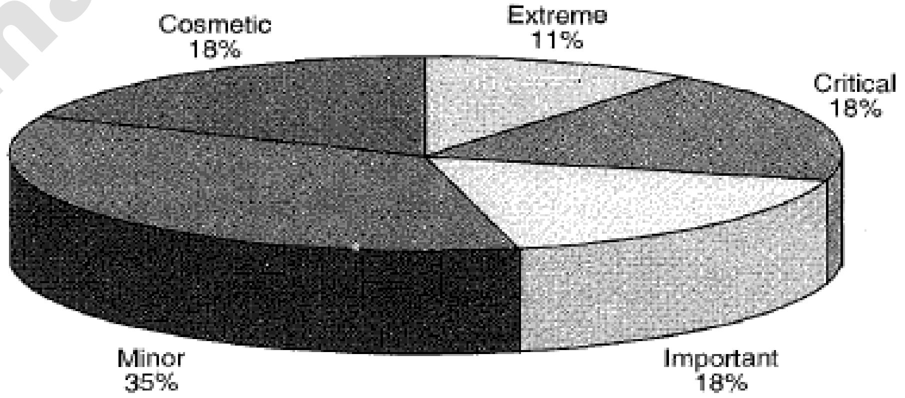
1. The find rate, fix rate, outstanding defects, and priority outstanding follow a bell curve pattern, indicating readiness for release at the end of the 19th week.
2. a sudden downward movement as well as upward spike in defect fixes rate needs analysis (13th to 17th week in the chart above)
3. By looking at the priority outstanding which shows close to zero defects in the 19th week, it can be concluded that all outstanding defects belong to low priority.
4. A smooth priority outstanding rate suggests that priority defects were closely tracked and fixed.

f) Defect Classification trend

Providing the perspective of defect classification in the chart helps in finding out release readiness of the product. When talking about the total number of outstanding defects, some of the questions that can be asked are

- How many of them are extreme defects?
- How many are critical?
- How many are important?

These questions require the charts to be plotted separately based on defect classification. The sum of extreme, critical, important, minor, and cosmetic defects is equal to the total number of defects.



Pie chart of defect distribution

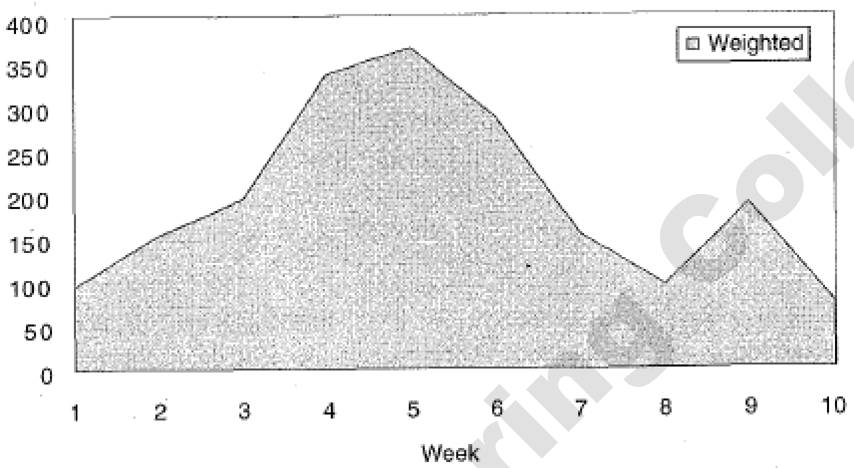
Arunai Engineering College

g) Weighted defects trend

Weighted defect helps in quick analysis of defect, instead of worrying about the classification of defects.

Weighted defects= (Extreme * 5 + Critical * 4 + important * 3 + Minor * 2 + Cosmetic)

Both "large defects" and "large number of small defects" affect product release.



Weighted defects trend.

From Figure it can be noted that

1. The ninth week has more weighted defects, which means existence of "large number of small defects" or "significant numbers of large defects" or a combination of the two. This is consistent with our interpretation of the same data using the stacked area chart.
2. The tenth week has a significant (more than 50) number of weighted defects indicating the product is not ready for release.

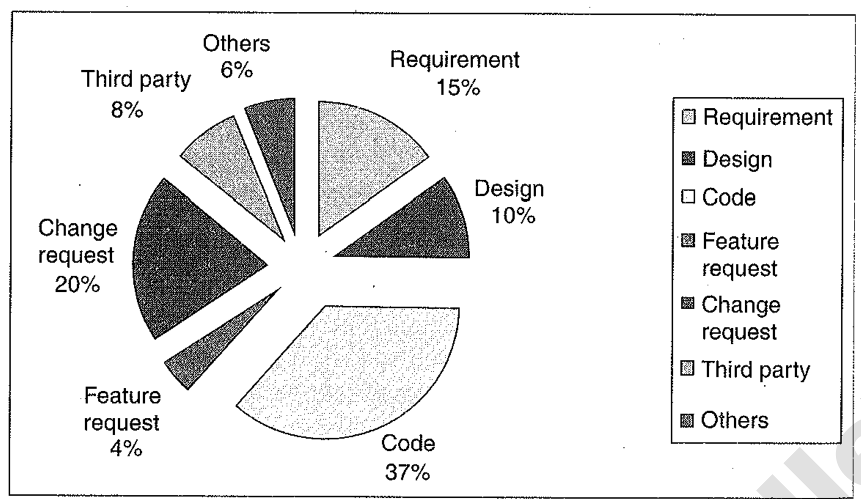
h) Defect cause distribution

Logical questions that would arise are:

1. Why are those defects occurring and what are the root causes?
2. What areas must be focused for getting more defects out of testing?

Finding the root causes of the defects help in identifying more defects and sometimes help in even preventing the defects.

Arunai Engineering College



Defect cause distribution chart

2.Development Defect Metrics

To map the defects to different components of the product , the parameter is LOC. It has

a) Component wise Defect Distribution
b) Defect Density & defect removal rate
c) Age Analysis of outstanding defect
d) Introduced and reopened defects trend

a) Component wise Defect Distribution

When module wise defect distribution is done , modules like install ,reports , client and database has > 20 defects indicating that more focus and resources are needed for these components.
So knowing the components producing more defects helps in defect fix plan and in deciding what to release.

b) Defect Density & defect removal rate

Defect density maps the defects in the product with the volume of code that is produced for the product.

Defects per KLOC = Total defects found in the product / total Executable line of code in KLOC

Variants to this metrics is to calculate AMD (add , modify , delete code) to find how a release affects product quality .

Defects per KLOC = Total defects found in the product / total Executable AMD line of code in KLOC

Defect removal rate = (defect found by verification activities + defects found in Unit test) / defect found by testing team) * 100

Arunai Engineering College

c) Age Analysis of outstanding defect

The time needed to fix a defect may be proportional to its age. It helps in finding out whether the defects are fixed as soon as they arrive and to ensure that long pending defects are given adequate priority.

d) Introduced and reopened defects trend

Introduced defect(ID): when adding new code or modifying the code to provide a defect fix , something that was working earlier may stop working , this is called ID.

reopened defects :fix that is provided in the code may not have fixed the problem completely or some other modification may have reproduced a defect that was fixed earlier. This is called as reopened defects.

Testing is not meant to find the same defects again ; release readiness should consider the quality of defect fixes.

III) PRODUCTIVITY METRICS

Productivity metrics combine several measurements and parameters with effort spend on the product. They help in finding out the capability of the team as well as for other purpose, such as

1. Estimating for the new release.
2. Finding out how well the team is progressing, understanding the reasons for (both positive and negative) variations in results.
3. Estimating the number of defects that can be found
4. Estimating release data and quality
5. Estimating the cost involved in the release.

a) Defects per 100 Hours of Testing

Defects per 100 hours of testing= (Total defects found in the product for a period / Total hours spent to get those defects) * 100

Test Cases Executed per 100 Hours of Testing

Test cases executed per 100 hours of testing = (Total test cases executed for a period / Total hours spent in test execution) * 100

b) Test cases Developed per 100 Hours of Testing

Test cases developed per 100 hours of testing= (Total test cases developed for a period / Total hours spent in test case development) * 100

c) Defects per 100 Test Cases

Defects per 100 test cases = (Total defects found for a period / Total test cases executed for the same period) * 100

Arunai Engineering College

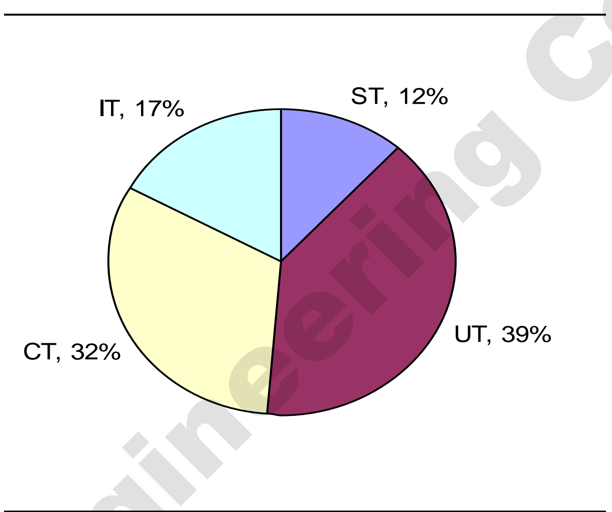
d) Defects per 100 Failed Test Cases

Defects per 100 failed test cases = (Total defects found for a period/Total test cases failed due to those defects) * 100

e) Test Phase Effectiveness

The following few observations can be made

1. A good proportion of defects were found in the early phases of testing (UT and CT).
2. Product quality improved from phase to phase (shown by less percent of defects found in the later test phases – IT and ST)



Test phase effectiveness

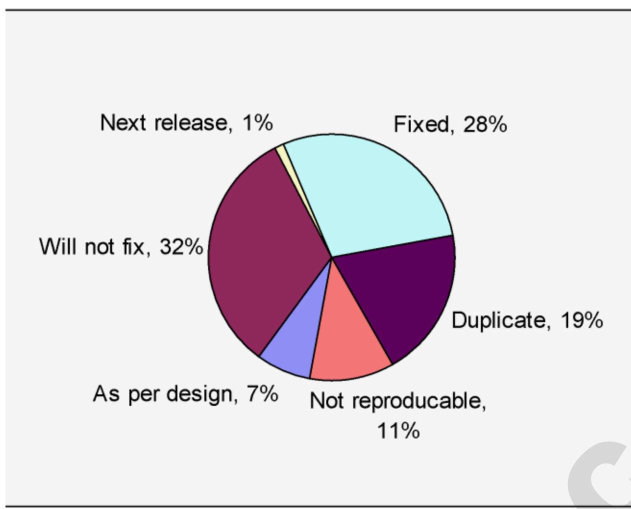
f) Closed Defect Distribution

The closed defect distribution helps in this analysis as shown in the figure below. From the chart, the following observations can be made.

1. Only 28% of the defects found by test team were fixed in the product. This suggests that product quality needs improvement before release.
2. Of the defects filled 19% were duplicates. It suggests that the test team needs to update itself on existing defects before new defects are filed.
3. Non-reproducible defects amounted to 11%. This means that the product has some random defects or the defects are not provided with reproducible test cases. This area needs further analysis.

Arunai Engineering College

4. Close to 40% of defects were not fixed for reasons “as per design,” “will not fix,” and “next release.” These defects may impact the customers. They need further discussion and defect fixes need to be provided to improve release quality.



Closed defect distribution

Arunai Engineering College