



Software Testing



How the customer explained it



How the Project Leader understood it



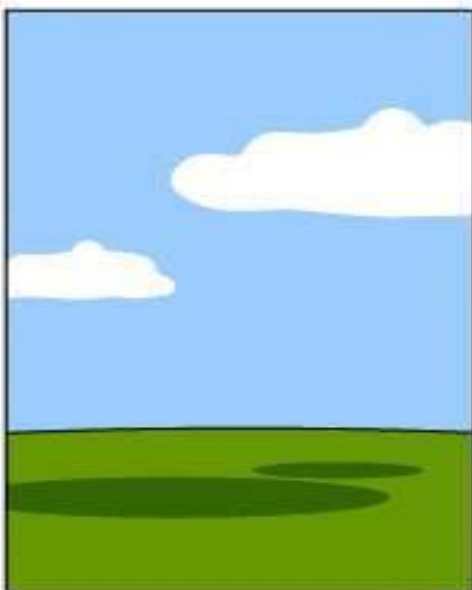
How the Analyst designed it



How the Programmer wrote it



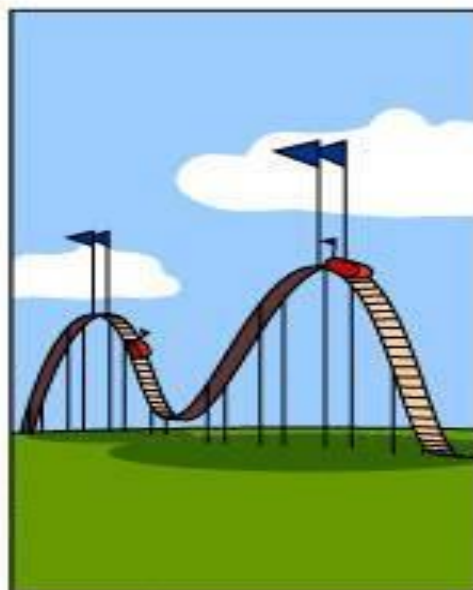
How the Business Consultant described it



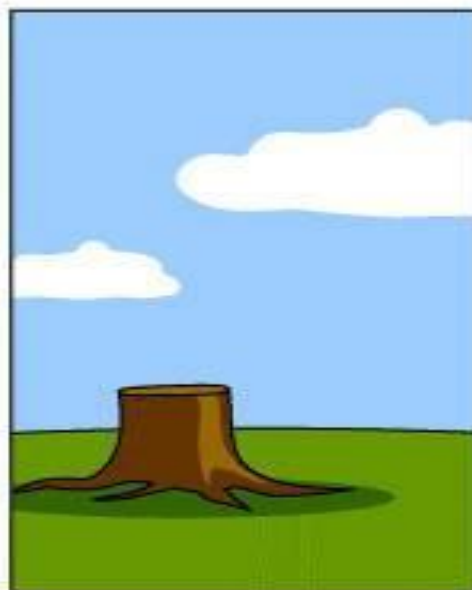
How the project was documented



What operations installed



How the customer was billed

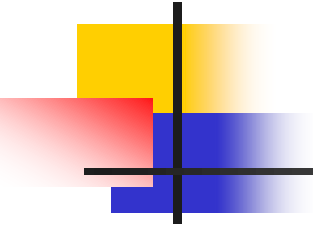


How it was supported



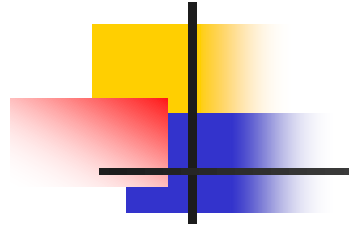
What the customer really needed

BACKGROUND



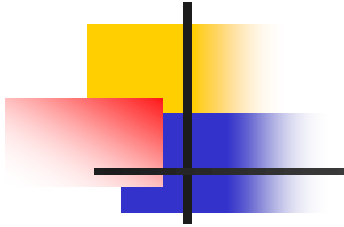
- Main objectives of a project: High Quality & High Productivity (Q&P)
- Quality has many dimensions
 - reliability, maintainability, interoperability etc.
- Reliability is perhaps the most important
- Reliability: The chances of software failing
- More defects => more chances of failure => lesser reliability
- Hence Q goal: Have as few defects as possible in the delivered software

TESTING OBJECTIVE



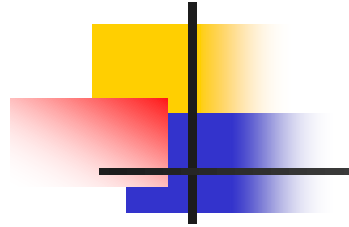
- Though errors are detected after each phase by techniques like inspections, some errors remain undetected.
- Final code is likely to have some requirements errors and design errors, in addition to errors introduced during the coding activity.
- Testing is the activity where the errors remaining from all the previous phases must be detected.
- During testing, the software to be tested is executed with a set of test cases.
- Behavior of the system for the test cases is evaluated to determine if the system is performing as expected.

ERROR, FAULT AND FAILURE



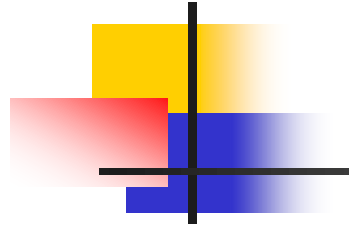
- Error : A human action that produces an incorrect result, Also referred to as mistake.
- Fault : Manifestation/presence of an error in software, also known as Defect or Bug.
- Failure : Deviation of the software from its expected/specified behavior

FAULTS & FAILURE



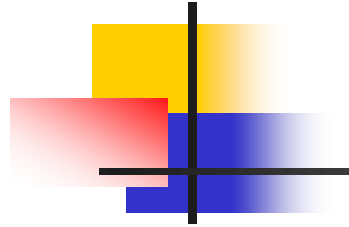
- Failure: A software failure occurs if the behavior of the s/w is different from expected/specified.
- A failure is produced only when there is a fault in the system
- Fault: cause of software failure
- Failure implies presence of defects
- A fault has the potential to cause failure.
- However, presence of a fault does not guarantee a failure

ROLE OF TESTING



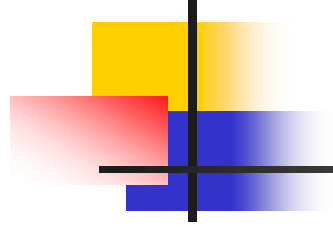
- Reviews are human processes - can not catch all defects
- Hence there will be requirement defects, design defects and coding defects in code
- These defects have to be identified by testing
- Therefore testing plays a critical role in ensuring quality.
- All defects remaining from before as well as new ones introduced have to be identified by testing.

TEST ORACLE



- To test any program, we need to have a description of its expected behavior
- A method of determining whether the actual behavior conforms to the expected behavior.
- For this we need a test oracle,
- Test oracle is a mechanism that determines whether software executed correctly for a test case.
- Testing is a process in which the test cases are given to the test oracle and the program under testing

TEST CASE



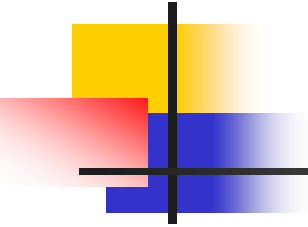
- A TEST CASE is a set of conditions under which a tester will determine whether a system under test satisfies requirements and works correctly
- Developing test cases can also help find problems in the requirements or design
- Test case sheet is designed with all possible test cases to do the exhaustive testing

TEST TITLE	PRIORITY	TEST CASE ID	TEST NUMBER	TEST DATE

TEST DESCRIPTION	TEST DEPENDENCIES	TEST CONDITIONS	TEST CONTROL

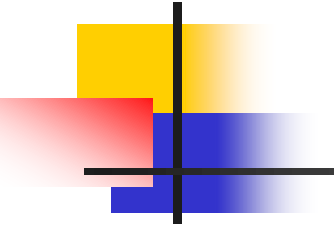
[illegible]

ROLE OF TEST CASES



- Having test cases that are good at revealing the presence of faults is central to successful testing
- If there is a fault in a program, the program can still provide the expected behavior for many inputs.
- Only for the set of inputs that exercise the fault in the program will the output of the program deviate from the expected behavior

TEST CASE CRITERIA

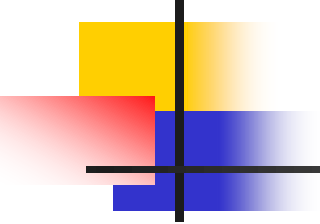


- Ideally, we would like to determine a set of test cases such that successful execution of all of them implies that there are no errors in the program
- Each test case costs money, as effort is needed to generate the test case
- Therefore, we would also like to minimize the number of test cases needed to detect errors
- Maximize the number of errors detected and minimize the number of test cases



Software Testing Techniques

Testability

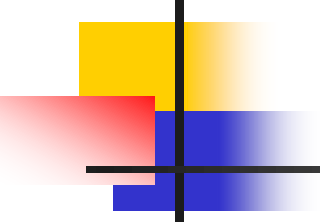
- 
-
- Software testability is simply how easily a computer program can be tested.
 - Testing must exhibit set of characteristics that achieve the goal of finding errors with a minimum of effort.

Testing attributes



1. A good test has a high probability of finding an error.
 - Tester must understand the software and attempt to develop a mental picture of how the software might fail.
2. A good test is not redundant.
 - Testing time and resources are limited.
 - There is no point in conducting a test that has the same purpose as another test.
 - Every test should have a different purpose
 - Ex. Valid/ invalid password.
3. A good test should be "best of breed"
 - In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests.
4. A good test should be neither too simple nor too complex.
 - sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors.
 - Each test should be executed separately

Test Case Design

- 
-
- Any engineered product (and most other things) can be tested in one of two ways:
 - Tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;
 - Tests can be conducted to ensure that all internal operations are performed according to specifications and all internal components have been adequately exercised.

Test cases and Test suites



- Test case is a triplet $[I, S, O]$ where
 - I is input data
 - S is state of system at which data will be input
 - O is the **expected** output
- Test suite is set of all test cases
- Test cases are not randomly selected. Instead even they need to be designed.

Test Data and Test Cases



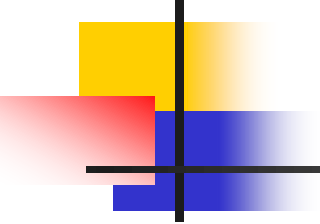
- ***Test data:*** Inputs which have been devised to test the system.
- ***Test cases:*** Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification.

Test-to-pass and test-to-fail

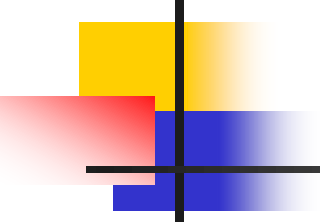


- Test-to-pass:
 - assures that the software minimally works,
 - does not push the capabilities of the software,
 - applies simple and straightforward test cases,
 - does not try to “break” the program.
- Test-to-fail:
 - designing and running test cases with the sole purpose of breaking the software.
 - strategically chosen test cases to probe for common weaknesses in the software.

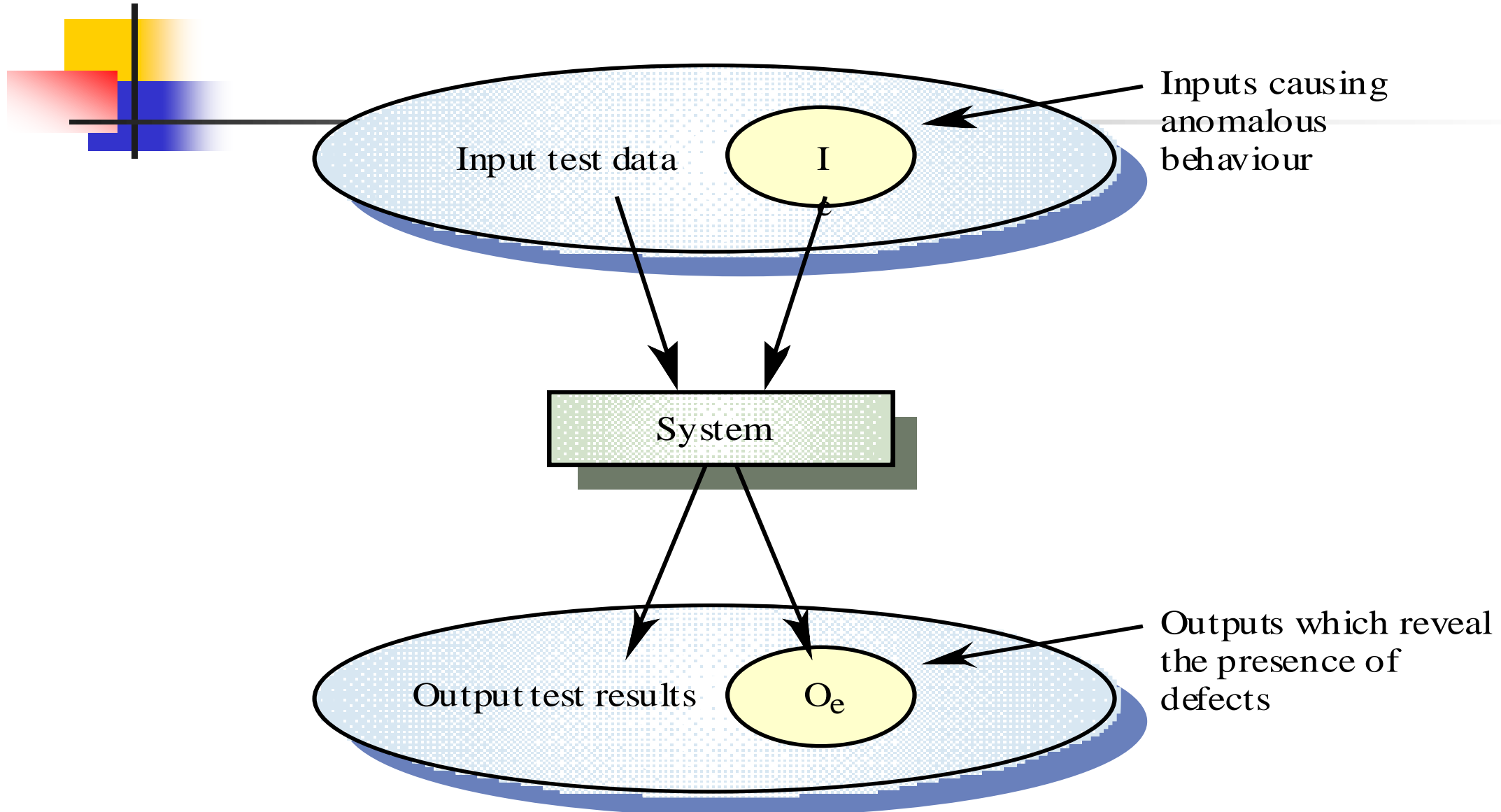
Black box testing

- 
- Also called *behavioral testing*, focuses on the functional requirements of the software.
 - It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
 - Black-box testing is not an alternative to white-box techniques but it is complementary approach.
 - Black-box testing attempts to find errors in the following categories:
 - Incorrect or missing functions,
 - Interface errors,
 - Errors in data structures or external data base access.
 - Behavior or performance errors,
 - Initialization and termination errors.

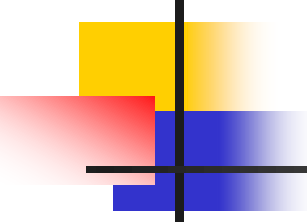
Black-box testing

- 
- Characteristics of Black-box testing:
 - Program is treated as a black box.
 - Implementation details do not matter.
 - Requires an end-user perspective.
 - Criteria are not precise.
 - Test planning can begin early.

Black-box testing

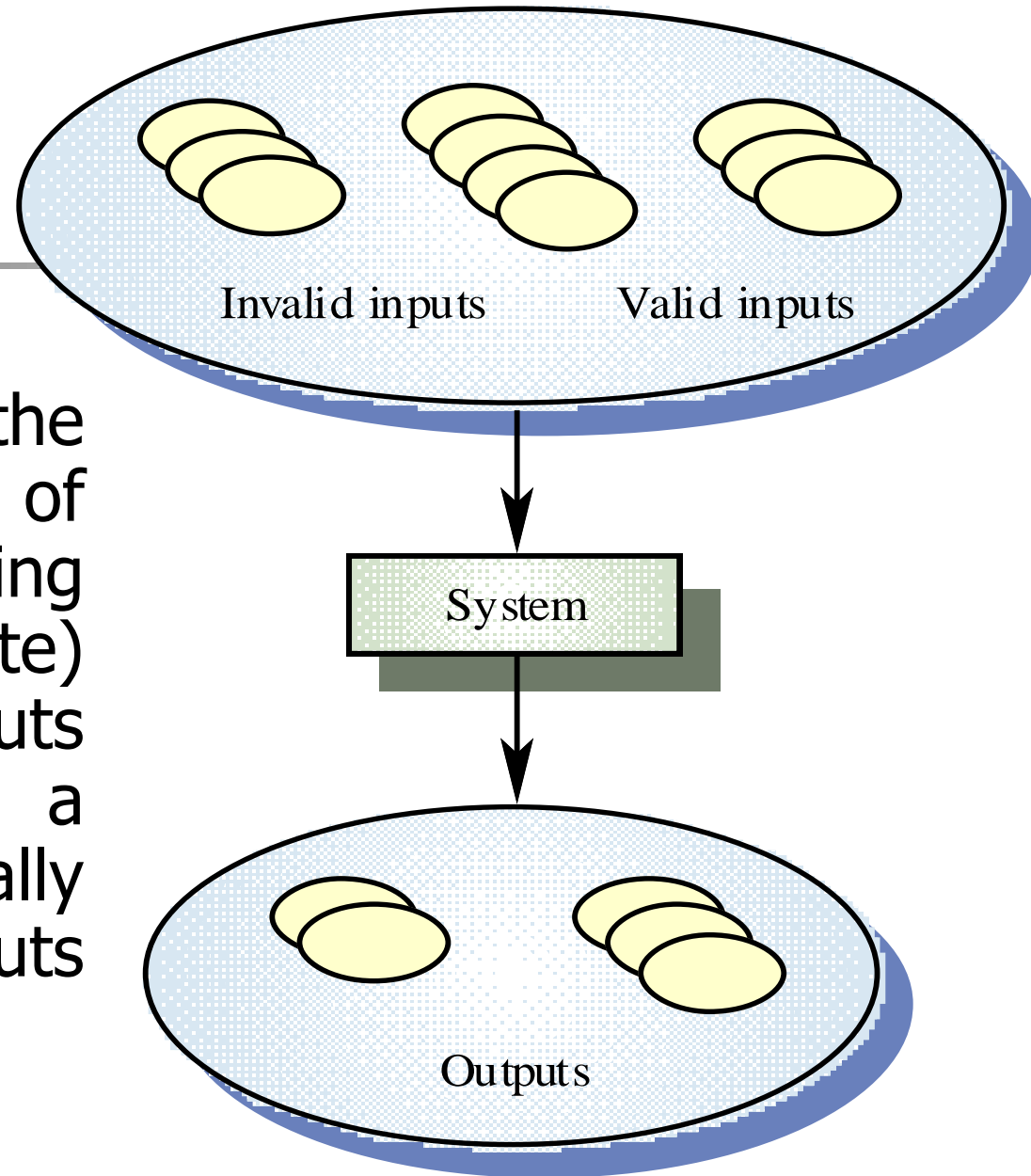


EQUIVALENCE CLASS PARTITIONING

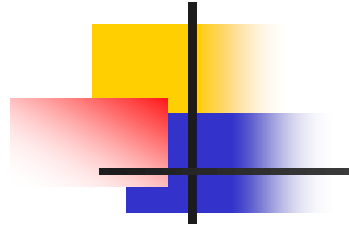
- 
- *Equivalence partitioning* is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
 - Test case design for equivalence partitioning is based on an evaluation of *equivalence classes* for an input condition.
 - An *equivalence class* represents a set of valid or invalid states for input conditions.
 - Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.
 - $\text{TestCase} = \{\text{Input}, \text{Operation}, \text{Output}\}$

Equivalence Partitioning

- Equivalence partitioning is the process of methodically reducing the huge (or infinite) set of possible inputs to test cases into a small, but equally effective, set of inputs for test cases.



Equivalence Classes for Range



If an input condition specifies a *range*, one valid and two invalid equivalence classes are defined.

AGE

Enter Age

*Accepts value 18 to 56

EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
≤ 17	18-56	≥ 57

Test Case for range

**Valid T1 {18-56, User Enter a value from 18 to 56, You can join govt services}

**Invalid T2 { ≤ 17 , User Enter value ≤ 17 , You can not join govt services}

**Invalid T3 { ≥ 57 , User Enter value ≥ 57 , You can not join govt services}

Equivalence Classes for Absolute Value

If an input condition requires a specific *value*, one valid and two invalid equivalence classes are defined.

MOBILE NUMBER *Must be 10 digits

EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
987654321	9876543210	98765432109

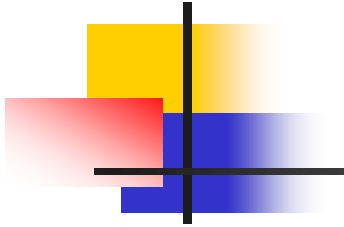
Test Case for absolute value

**Valid T1 {10 digits 0-9, User enters 10 digit comprising of 0 to 9, Valid mobile no. entered}

**Invalid T2 {≤9, User enters ≤9 digit comprising of 0 to 9, Invalid mobile no. is entered}

**Invalid T3 {≥10, User enters ≥11, Invalid mobile no. is entered}

Equivalence Classes for set of Value



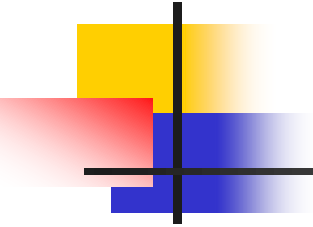
If an input condition requires for set of *values*, one valid and one invalid equivalence classes are defined.

For example we have to take details of student where attendance is (75, 85, 95), (Punjab, Haryana, New Delhi)

Test Case for set of values

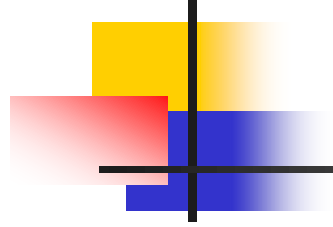
****Valid T1** {(Any value of 75,85,95), User enters a value like 75,85,95,(Details to be displayed on the screen)}

****Invalid T2** {(All values other than 75,85,95), User enters value other than 75,85,95,(Details are not available for input entered)}



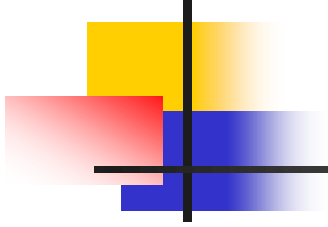
1. If an input condition requires a absolute *value*, one valid and two invalid equivalence classes are defined.
2. If an input condition specifies a member of a *set*, one valid and one invalid equivalence class are defined.
3. If an input condition is *Boolean*, one valid and one invalid class are defined.

BOUNDARY VALUE ANALYSIS (BVA)



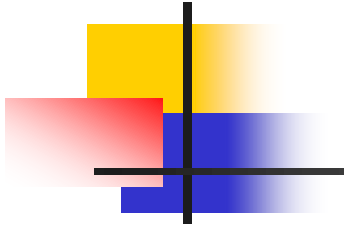
- Boundary value analysis is a test case design technique that complements equivalence class partitioning.
- Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class.
- In other word, Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

BOUNDARY VALUE ANALYSIS



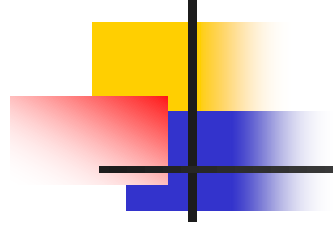
- Some typical programming errors occur:
 - At boundaries of equivalence classes
- Programmers often fail to see:
 - Special processing required at the boundaries of equivalence classes.

BOUNDARY VALUE ANALYSIS



- Programmers may improperly use $<$ instead of $<=$
- Boundary value analysis:
 - select test cases at the boundaries of different equivalence classes.

GUIDELINES FOR BVA



1. If an input condition specifies a range bounded by values a and b , test cases should be designed with values a and b and just above and just below a and b .
2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
3. Apply guidelines 1 and 2 to output conditions.

BVA Example

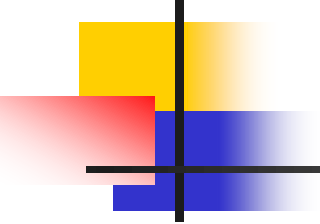
- For a function that computes the square root of an integer in the range of 1 and 5000:
 - test cases must include the values: {0,1,5000,5001}.
 - TC1{0,1,5000,5001}





White Box Testing

White Box Design

- 
-
- Also called as glass-box testing.
 - Uses the control structure of the procedural design to derive test cases.
 - Using white-box testing we can derive test cases that
 - Guarantee that all independent paths within a module have been exercised at least once,
 - Exercise all logical decisions on their true and false sides,
 - Execute all loops at their boundaries and within their operational bounds,
 - Exercise internal data structures to ensure their validity.

White Box Design

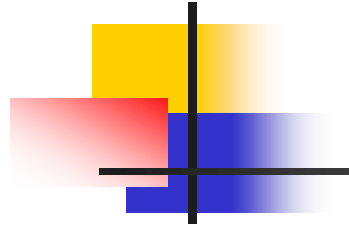


- "Why spend time and energy worrying about (and testing) logics when we might better expend effort ensuring that program requirements have been met?"

OR

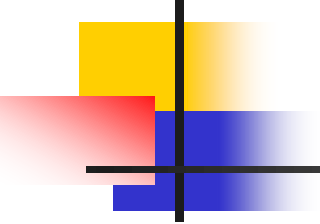
- Why don't we spend all of our energy on black-box tests?

White Box Design



- The answer lies in the nature of software defects:
 - Logical errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed
 - We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis.
 - Typographical errors are random
- Each of these reasons provides an argument for conducting white-box tests.

Code Coverage Testing



Since a product is realized in terms of program code, if we can run test cases to exercise the different parts of the code, then that part of the product realized by the code gets tested. Code coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing. The percentage of code covered by a test is found by adopting a technique called *instrumentation* of code. There are specialized tools available to achieve instrumentation. Instrumentation rebuilds the product, linking the product with a set of libraries provided by the tool

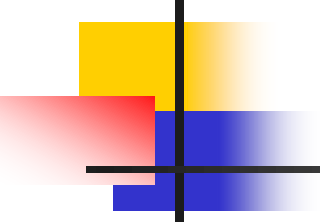
Code coverage testing is made up of the following types of coverage.

1. Statement coverage
2. Path coverage
3. Condition coverage
4. Function coverage

Statement coverage

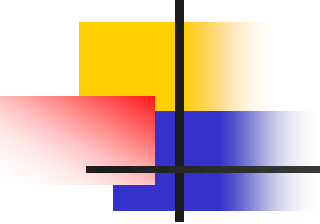


- Programs constructs can be usually classified as:
 1. Sequential Control Flow
 2. Two way decision statements (if then else)
 3. Multi way decision statements (switch)
 4. Loops (while, do, repeat until, for)

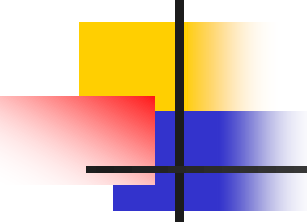
- 
- Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once.
 - It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.
 - Statement coverage is used to derive scenario based upon the structure of the code under test.

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

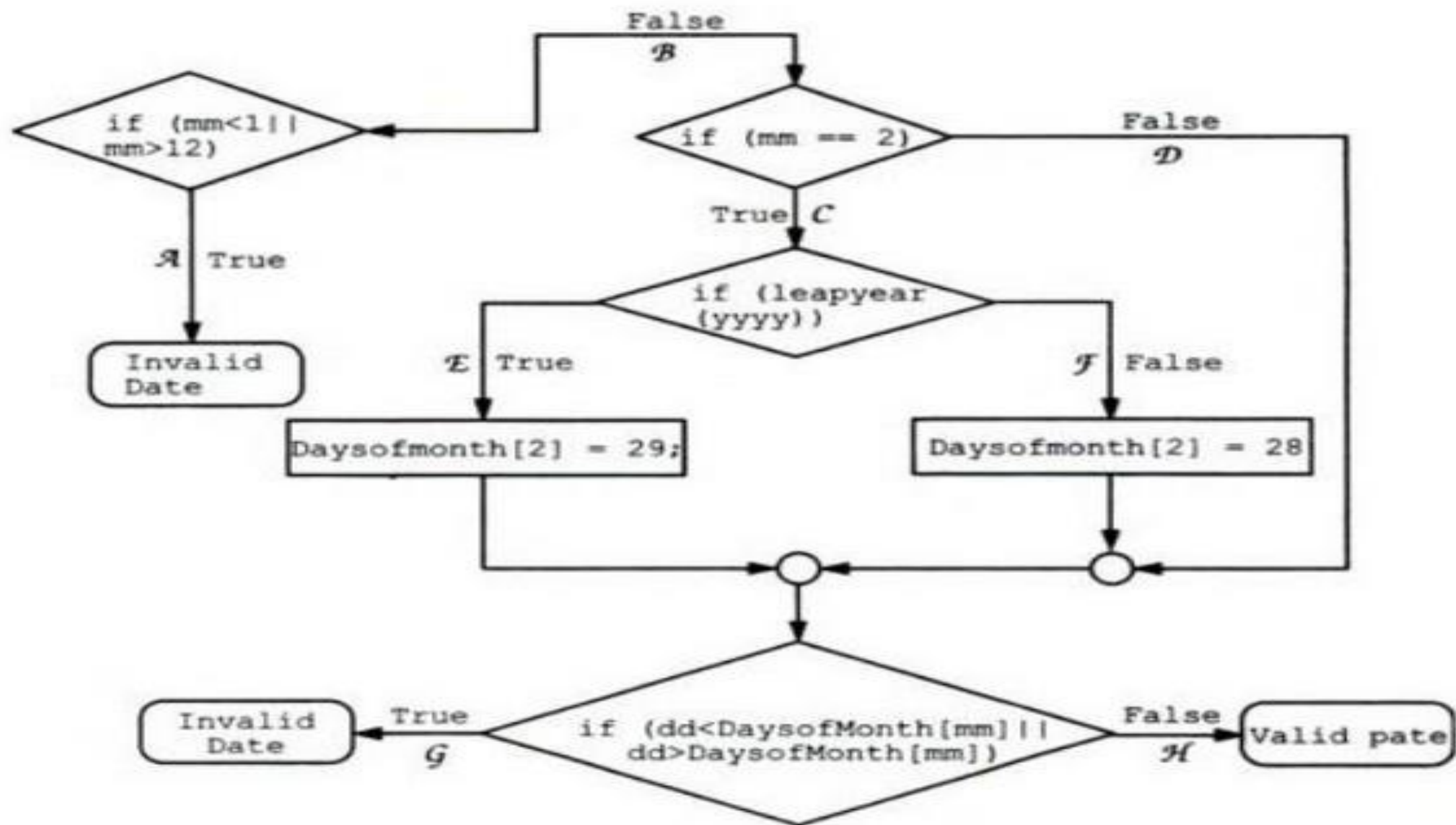
Path Coverage Testing

- 
- Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path.
 - It helps to determine all faults lying within a piece of code.
 - This method is designed to execute all or selected path through a computer program.

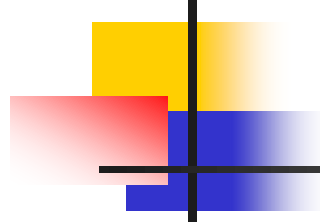
$$\text{Path Coverage} = \frac{\text{Total paths exercised / Total number of paths in program}}{1} \times 100$$



Let us take an example of a date validation routine. The date is accepted as three fields mm, dd and yyyy. We have assumed that prior to entering this routine, the values are checked to be numeric. To simplify the discussion, we have assumed the existence of a function called `leapyear` which will return TRUE if the given year is a leap year. There is an array called `DayofMonth` which contains the number of days in each month. A simplified flow chart



Condition Coverage



- Conditional coverage or expression coverage will reveal how the variables or subexpressions in the conditional statement are evaluated.
- In this coverage expressions with logical operands are only considered.
- Conditional coverage offers better sensitivity to the control flow than decision coverage.
- Condition coverage does not give a guarantee about full decision coverage

$$\text{Condition Coverage} = \frac{\text{Number of Executed Operands}}{\text{Total Number of Operands}} * 100$$

BLACK BOX TESTING

It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.

It is mostly done by software testers.

No knowledge of implementation is needed.

It can be referred as outer or external software testing.

WHITE BOX TESTING

It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.

It is mostly done by software developers.

Knowledge of implementation is required.

It is the inner or the internal software testing.

It is functional test of the software.

It is structural test of the software.

This testing can be initiated on the basis of requirement specifications document.

This type of testing of software is started after detail design document.

No knowledge of programming is required.

It is mandatory to have knowledge of programming.

It is the behavior testing of the software.

It is the logic testing of the software.

It is applicable to the higher levels of testing of software.

It is generally applicable to the lower levels of software testing.

It is also called closed testing.

It is also called as clear box testing.

It is least time consuming.

It is most time consuming.

It is not suitable or preferred for algorithm testing.

It is suitable for algorithm testing.

Can be done by trial and error ways and methods.

Data domains along with inner or internal boundaries can be better tested.



LEVELS OF TESTING

LEVELS OF TESTING

1

Unit Testing

Done by Developers

2

Integration Testing

Done by Testers

3

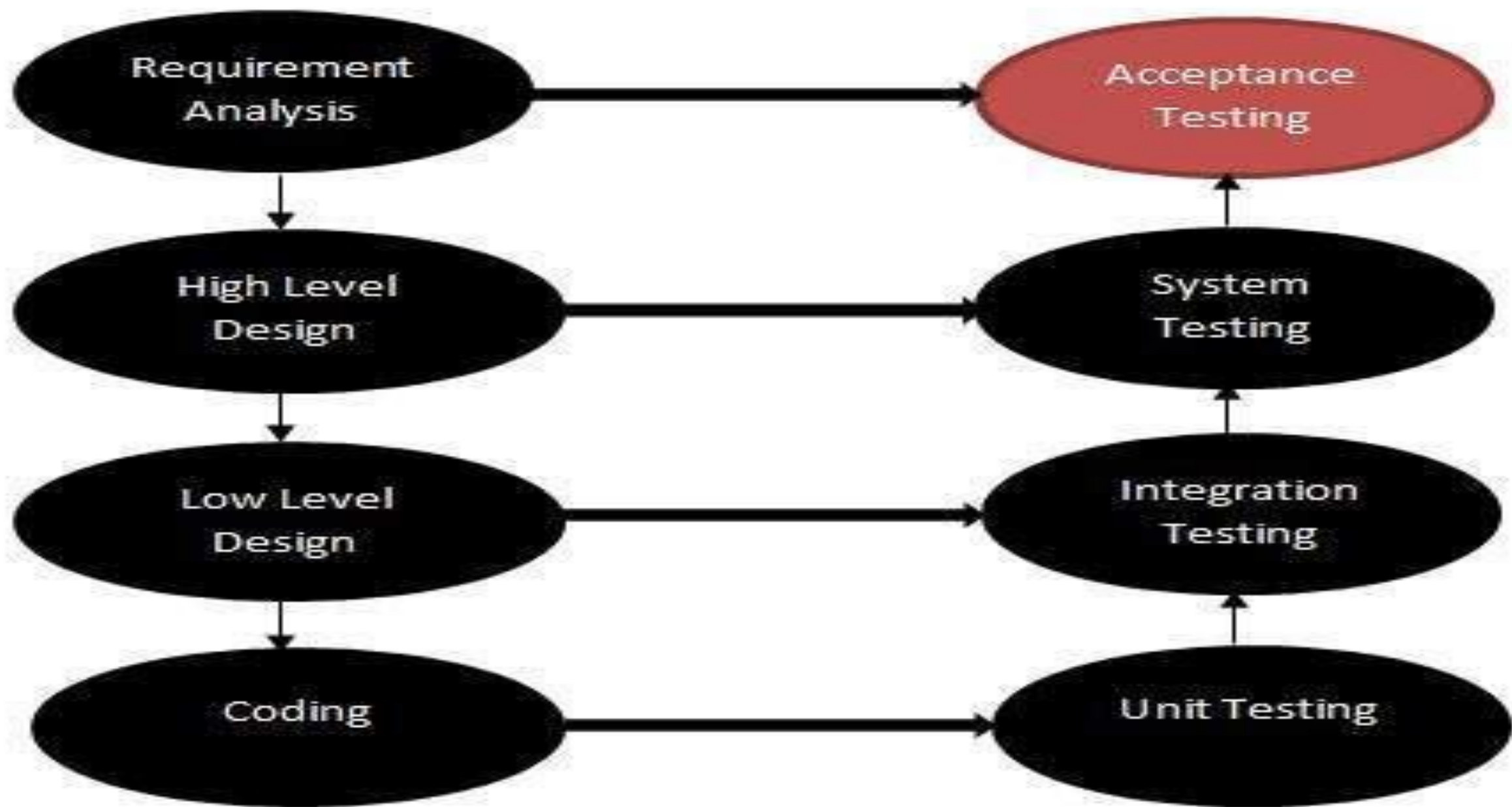
System Testing

Done by Testers

4

Acceptance Testing

Done by End Users

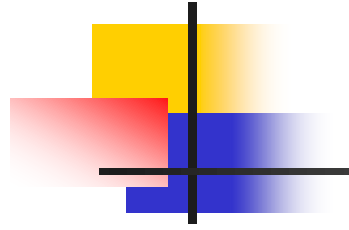


Unit Testing

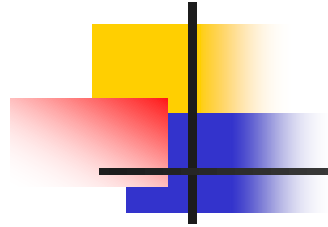


- Unit Testing is done to check whether the individual modules of the source code are working properly.
- Testing each and every unit of the application separately by the developer in the developer's environment.
- It is also known as Module Testing or Component Testing

Integration Testing

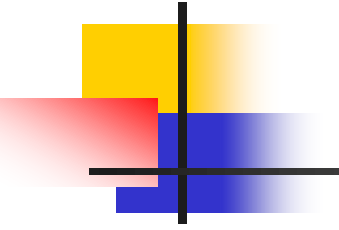


- Integration Testing is the process of testing the connectivity or data transfer between a couple of unit tested modules.
- It is subdivided into the Top-Down Approach, Bottom-Up Approach and Sandwich Approach (Combination of Top Down and Bottom Up).
- This process is carried out by using dummy programs called Stubs and Drivers.
- Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.
- Big Bang Integration Testing
 - In Big Bang Integration Testing, the individual modules are not integrated until all the modules are ready.
 - Then they will run to check whether it is performing well.
 - Defects can be found at the later stage which is its major disadvantage also.

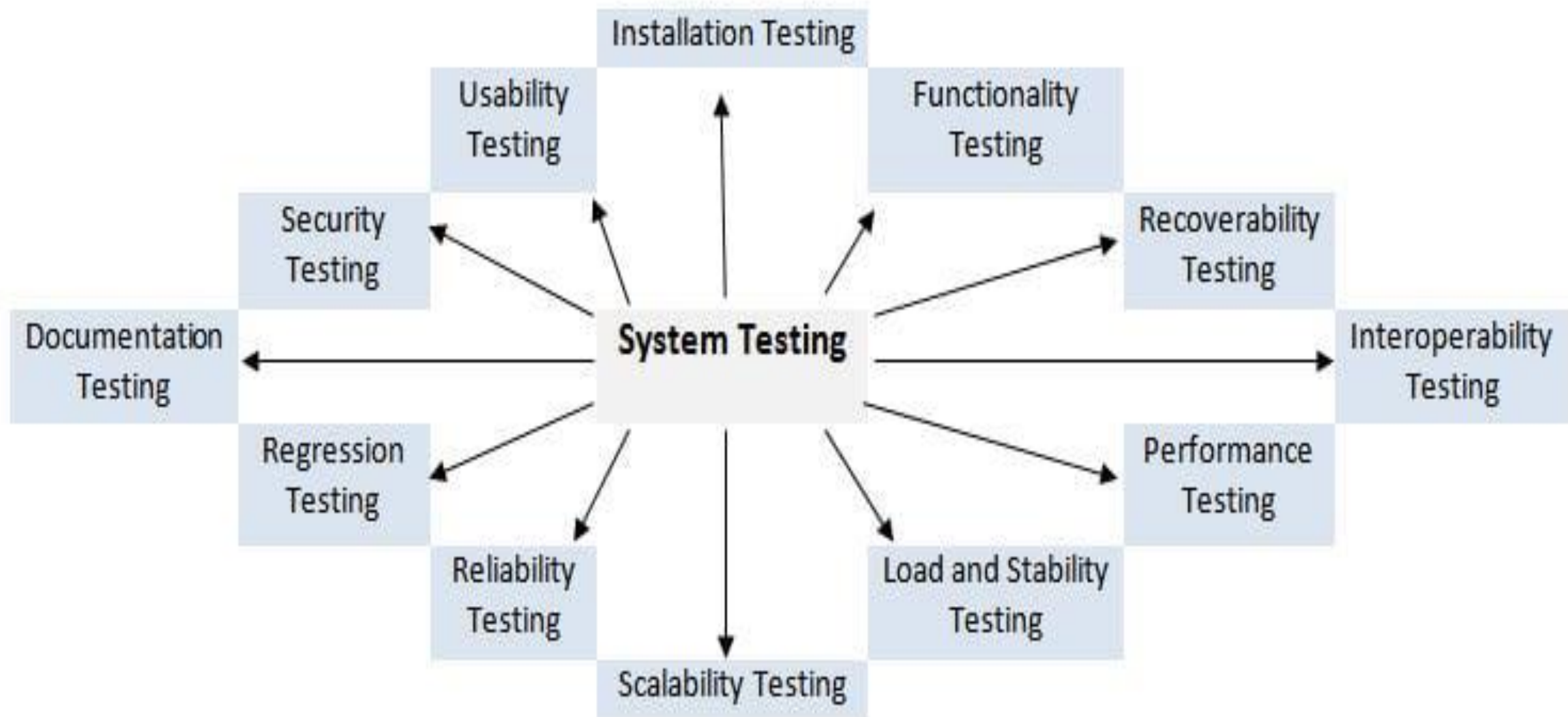


- Top Down Integration Testing
 - The high-level modules are integrated and tested first like Testing from the main module to the submodule.
 - Stubs are used as a temporary module if a module is not ready for integration testing.
- Bottom Up Integration Testing
 - In Bottom Up Integration Testing, the low-level modules are integrated and tested first like Testing from sub-module to the main module.
 - Same like Stubs, here drivers are used as a temporary module for integration testing.

System Testing

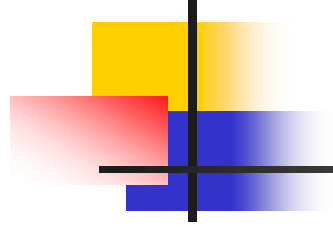


- It's a black box testing.
- Testing of fully integrated application this is also called as an end to end scenario testing.
- To ensure that the software works in all intended target systems. Validate thorough testing of every input in the application to check for desired outputs.
- Testing of the users' experiences with the application.

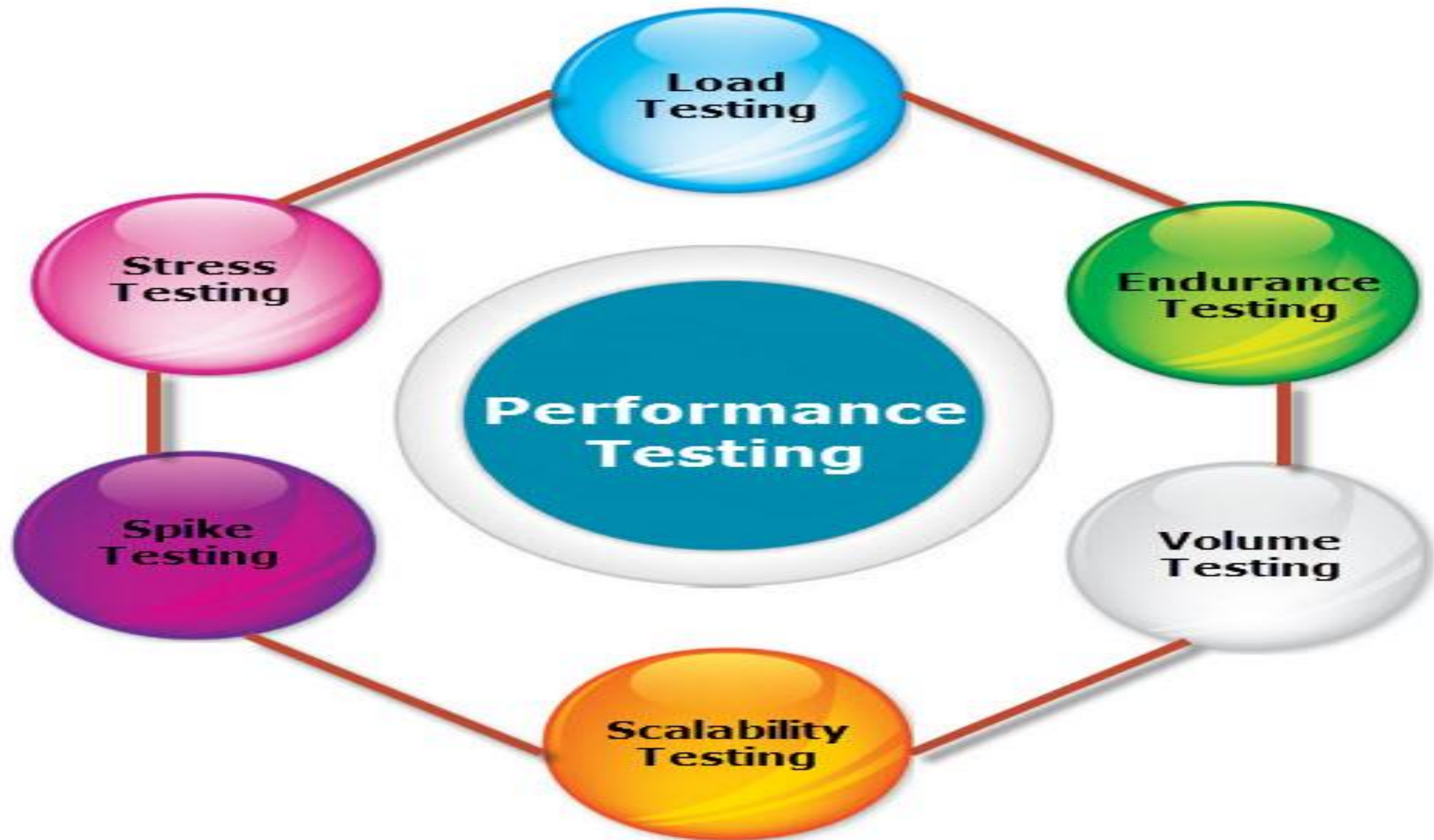


System Testing - © www.SoftwareTestingHelp.com

Performance Testing



- Software Performance testing is type of testing performed to determine the performance of system to measure, validate or verify quality attributes of the system like responsiveness, Speed, Scalability, Stability under variety of load conditions.
- Software performance testing involves the testing of application under test to ensure that application is working as expected under variety of load conditions.
- The goal of performance testing is not only find the bugs in the system but also eliminate the performance bottlenecks from the system.



■ **Load Testing:**

- To check system with constantly increasing the load on the system until the time load reaches to its threshold value.
- Increasing load means increasing number of concurrent users, transactions & check the behavior of application under test.
- Examples like testing printer by sending large job, running multiple applications simultaneously on server, continuously reading and writing data into hard disk etc.

■ **Stress Testing:**

- To check the stability of software when hardware resources are not sufficient like CPU, memory, disk space etc.
- To determine the failure of system and to keep an eye on how the system gracefully get recover back, this quality is known as recoverability.
- This testing is also known as **Fatigue testing**, this testing should capture the stability of the application by testing it beyond its bandwidth capacity.

■ **Spike Testing:**

- Spike testing is subset of Stress Testing.
- Carried out to validate the performance characteristics when the system is subjected to workload models and load volumes that repeatedly increase beyond anticipated production operations for short periods of time.

■ **Endurance Testing**

- Endurance testing involves testing a system with a expected amount of load over a long period of time to find the behavior of system.
- Let's take a example where system is designed to work for 3 hours of time but same system endure for 6 hours of time to check the staying power of system.
- Most commonly test cases are executed to check the behavior of system like memory leaks or system fails or random behavior.
- Sometimes endurance testing is also referred as Soak testing.

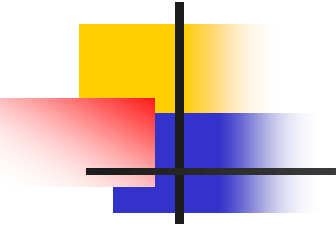
■ **Scalability Testing**

- Under Scalability software application is testing to determine its capability to scale up in terms of any of its non-functional capability like the user load supported, the number of transactions, the data volume etc.
- The main aim of this testing is to understand at what peak the system prevent more scaling.

■ **Volume Testing**

- Volume testing is non-functional testing which refers to testing a software application with a large amount of data to be processed to check the efficiency of the application.
- The main goal of this testing is to monitor the performance of application under varying database volumes.

Acceptance Testing



- A testing technique performed to determine whether or not the software system has met the requirement specifications.
- The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

■ **Alpha Testing:**

- Alpha testing is mostly like performing usability testing which is done by the in-house developers who developed the software.
- Sometimes this alpha testing is done by the client or outsiders with the presence of developers or testers.

■ **Beta Testing:**

- Beta testing is done by a limited number of end users before delivery, the change request would be fixed if the user gives feedback or reports defect.

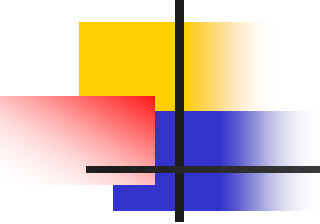
■ **Gamma Testing:**

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

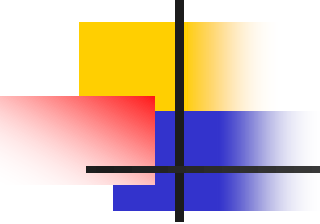


SOFTWARE TESTING PROCESS

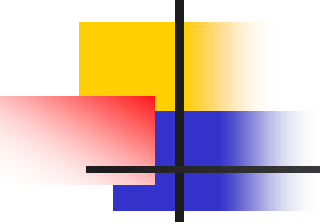
TEST PLAN

- 
- Testing usually starts with test plan and ends with acceptance testing.
 - Test plan is a general document that defines the scope and approach for testing for the whole project
 - Inputs are SRS, project plan, design documents etc.
 - Test plan identifies what levels of testing will be done, what units will be tested in the project.

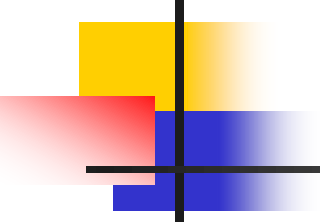
TEST PLAN...

- 
- Test plan usually contains
 - Test unit specs: what units need to be tested separately.
 - Features to be tested: these may include functionality, performance, usability.
 - Approach: criteria to be used, when to stop, when to resume, how to evaluate, etc.
 - Test deliverables
 - Schedule and task allocation

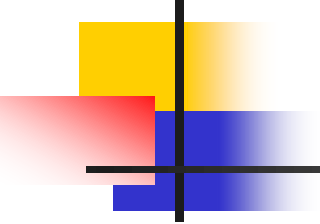
TEST CASE SPECIFICATIONS

- 
- Test plan focuses on approach; does not deal with details of testing a unit
 - Test case specification has to be done separately for each unit
 - Based on the plan (approach, features,..) test cases are determined for a unit
 - Expected outcome also needs to be specified for each test case

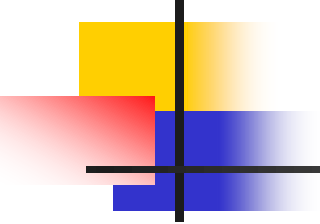
TEST CASE SPECIFICATIONS...

- 
- Together the set of test cases should detect most of the defects
 - Would like the set of test cases to detect any defects, if it exists
 - Would also like set of test cases to be small - each test case consumes effort
 - Determining a reasonable set of test case is the most challenging task of testing

TEST CASE SPECIFICATIONS...

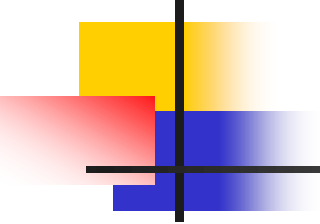
- 
- The effectiveness and cost of testing depends on the set of test cases
 - How to determine if a set of test cases is good? I.e. the set will detect most of the defects, and a smaller set cannot catch these defects
 - No easy way to determine goodness; usually the set of test cases is reviewed by experts
 - This requires test cases be specified before testing – a key reason for having test case specs
 - Test case specs are essentially a table

TEST CASE SPECIFICATIONS...

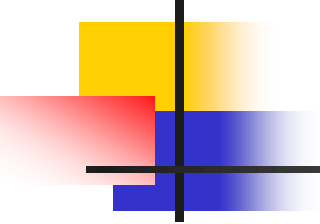


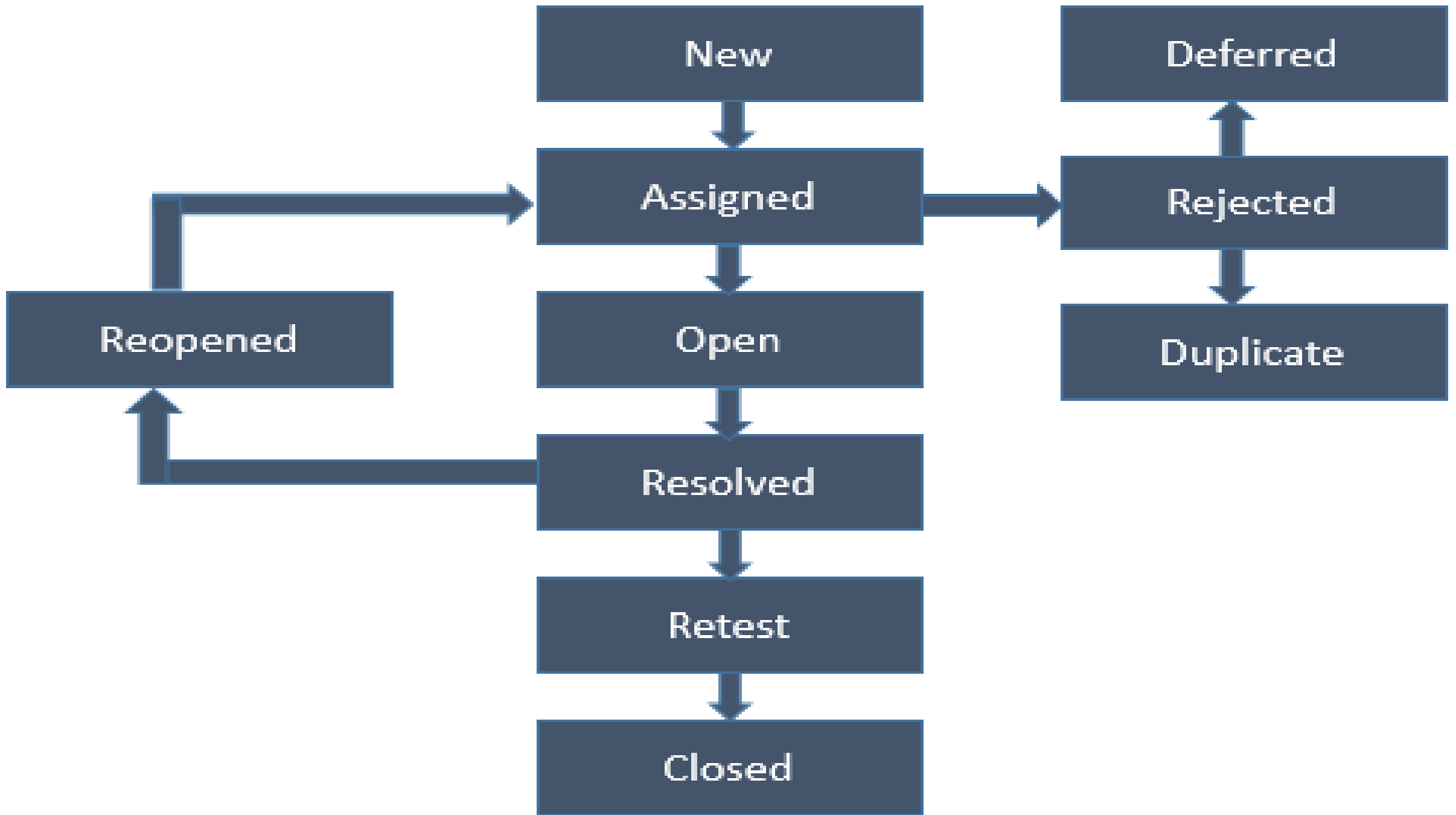
Seq.No	Condition to be tested	Test Data	Expected result	successful

TEST CASE SPECIFICATIONS...

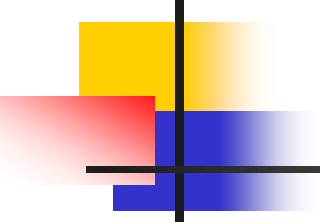
- 
- So for each testing, test case specs are developed, reviewed, executed and documented.
 - Preparing test case specifications is challenging and time consuming
 - Test case criteria can be used
 - Special cases and scenarios may be used
 - Once specified, the execution and checking of outputs may be automated through scripts
 - Desired if repeated testing is needed
 - Regularly done in large projects

TEST CASE EXECUTION AND ANALYSIS

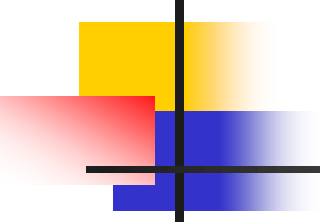
- 
- Executing test cases may require drivers or stubs to be written; some tests can be auto, others manual
 - A separate test procedure document may be prepared
 - Test summary report is often an output which gives a summary of test cases executed, effort, defects found, etc.
 - Monitoring of testing effort is important to ensure that sufficient time is spent.
 - Computer time also is an indicator of how testing is proceeding.



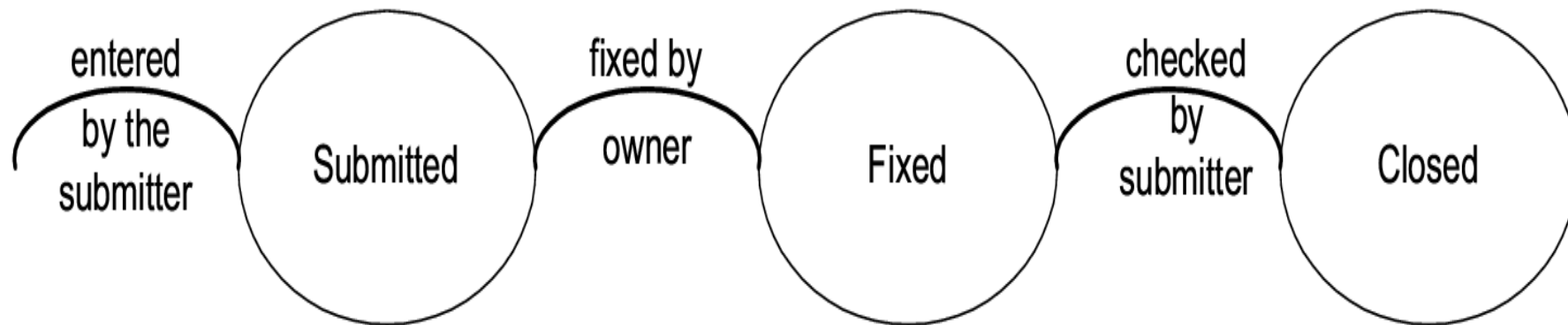
DEFECT LOGGING AND TRACKING

- 
- A large software may have thousands of defects, found by many different people
 - Often person who fixes (usually the coder) is different from who finds
 - Due to large scope, reporting and fixing of defects cannot be done informally
 - Defects found are usually logged in a defect tracking system and then tracked to closure
 - Defect logging and tracking is one of the best practices in industry

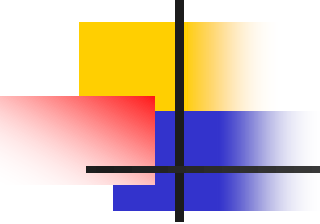
DEFECT LOGGING...

- 
- A defect in a software project has a life cycle of its own, like
 - Found by someone, sometime and logged along with info about it (submitted)
 - Job of fixing is assigned; person debugs and then fixes (fixed)
 - The manager or the submitter verifies that the defect is indeed fixed (closed)
 - More elaborate life cycles possible

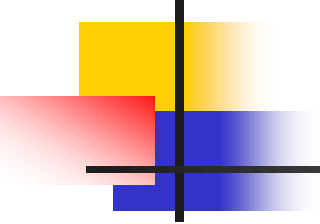
DEFECT LOGGING...



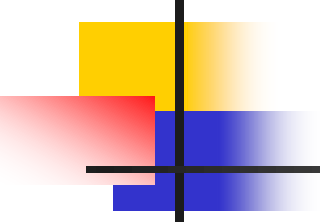
DEFECT LOGGING...

- 
- During the life cycle, info about defect is logged at diff stages to help debug as well as analysis
 - Defects generally categorized into a few types, and type of defects is recorded
 - Some std categories: Logic, standards, UI, interface, performance, documentation,..

DEFECT LOGGING...

- 
- Severity of defects in terms of its impact on software is also recorded
 - Severity is useful for prioritization of fixing
 - One categorization
 - Critical: Show stopper
 - Major: Has a large impact
 - Minor: An isolated defect
 - Cosmetic: No impact on functionality

DEFECT LOGGING AND TRACKING...

- 
- Ideally, all defects should be closed
 - Sometimes, organizations release software with known defects (hopefully of lower severity only)
 - Organizations have standards for when a product may be released
 - Defect log may be used to track the trend of how defect arrival and fixing is happening