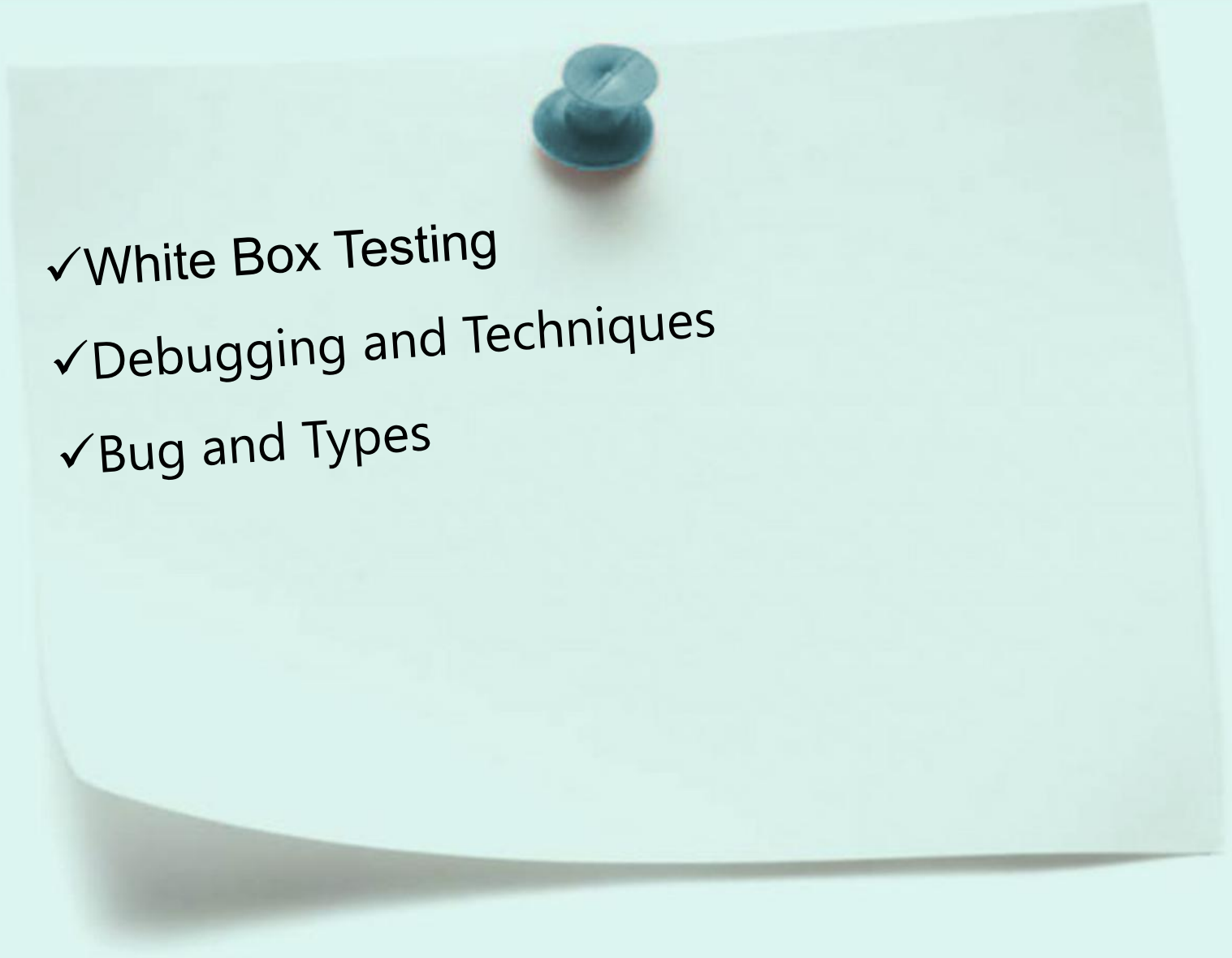
Software Engineering -1

Engr. M. Fahad Khan
Lecturer, Department Software Engineering
University of Engineering & Technology, Taxila

Topics To Be Covered Today

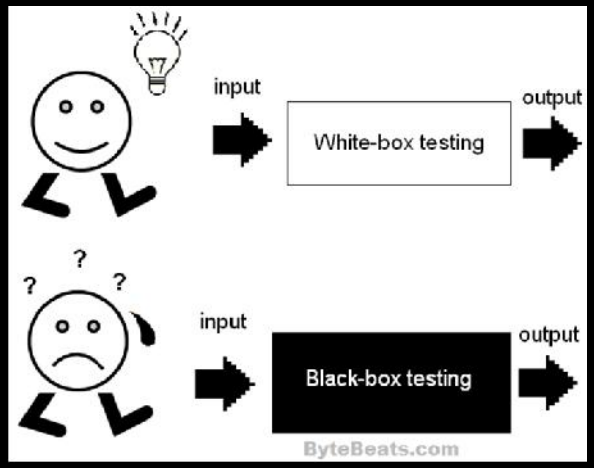


- ✓White Box Testing
- ✓Debugging and Techniques
- ✓Bug and Types

White Box Testing

White box testing (a.k.a. clear box testing, glass box testing, transparent box testing, translucent box testing or structural testing) .

The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc.
In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code.
White box test also needs the tester to look into the code and find out which unit/statement/chunk of the code is malfunctioning.



Synonyms for Glass Box Testing:

Glass-box testing is also called:

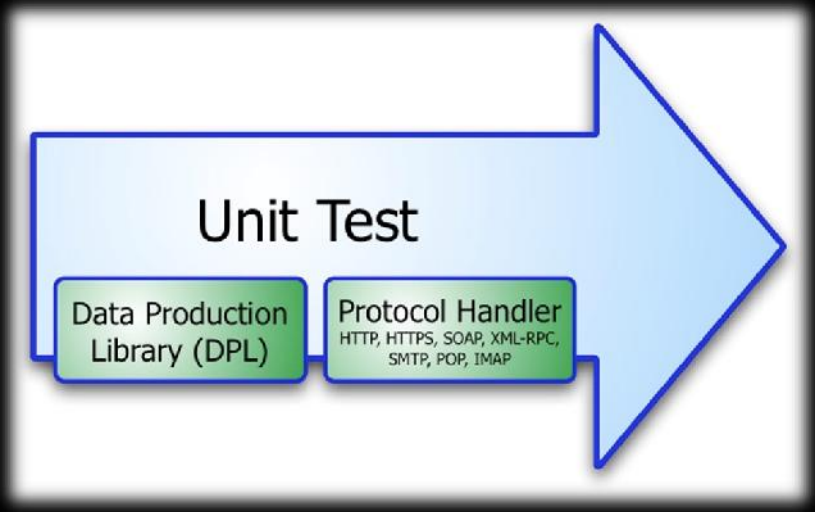
- ✓Clear Box testing
- ✓Structural testing
- ✓Open Box Testing
- ✓White Box testing

White Box Testing Techniques

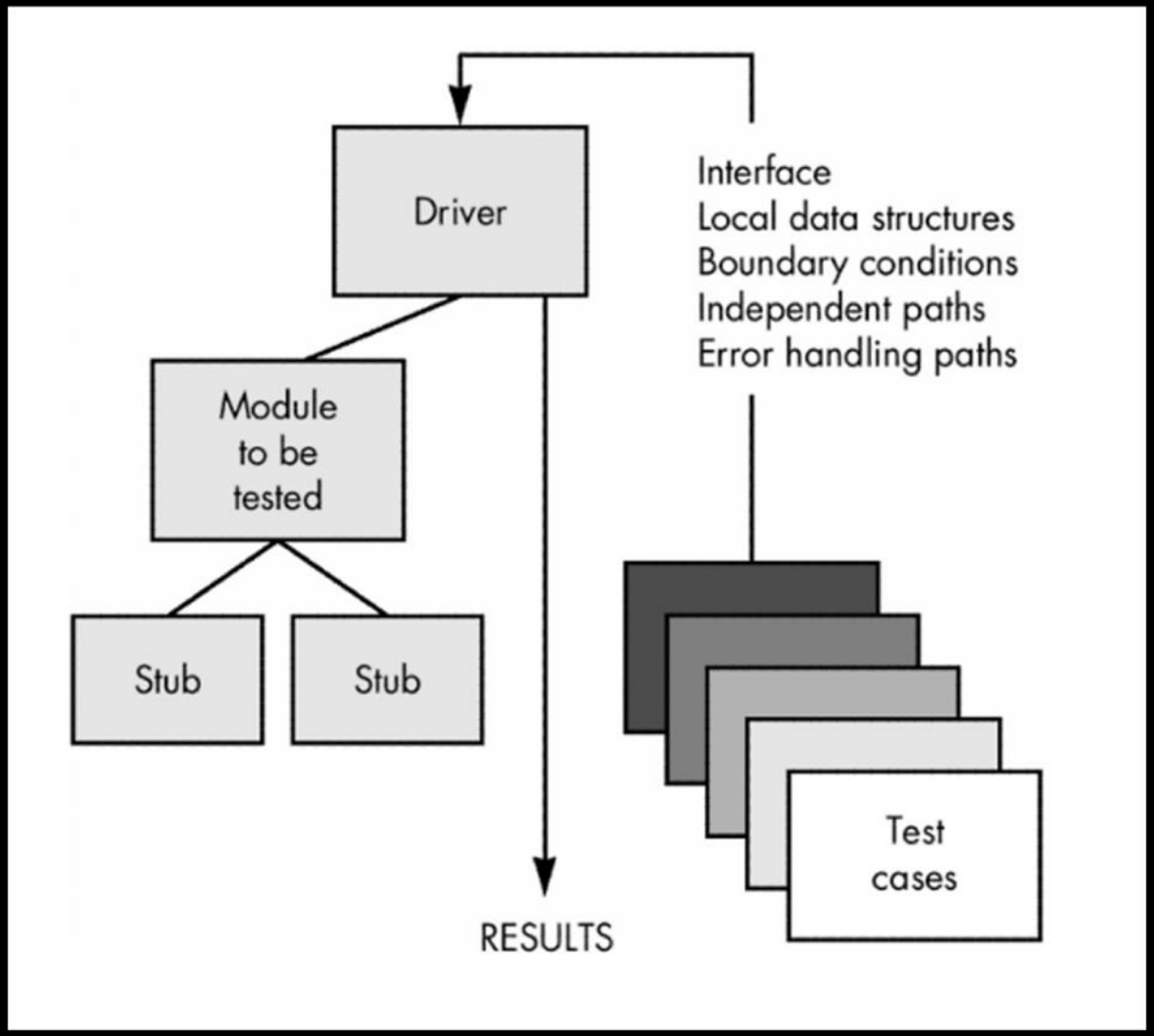
- Unit Testing
- Code Coverage and its Types
- Cyclomatic Complexity
- Graph matrices
- Basis Path Testing
- Control Flow Testing Examples
- Condition Testing
- Loop Testing
- Data Flow Testing
- Data Flow Coverage Criteria
- Terms used in data flow testing
- Data Flow Coverage Concept
- DC Path

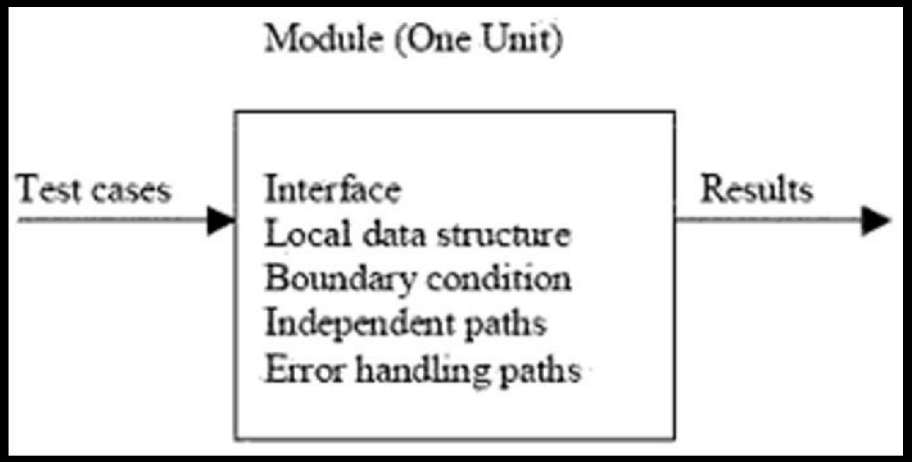
Unit Testing

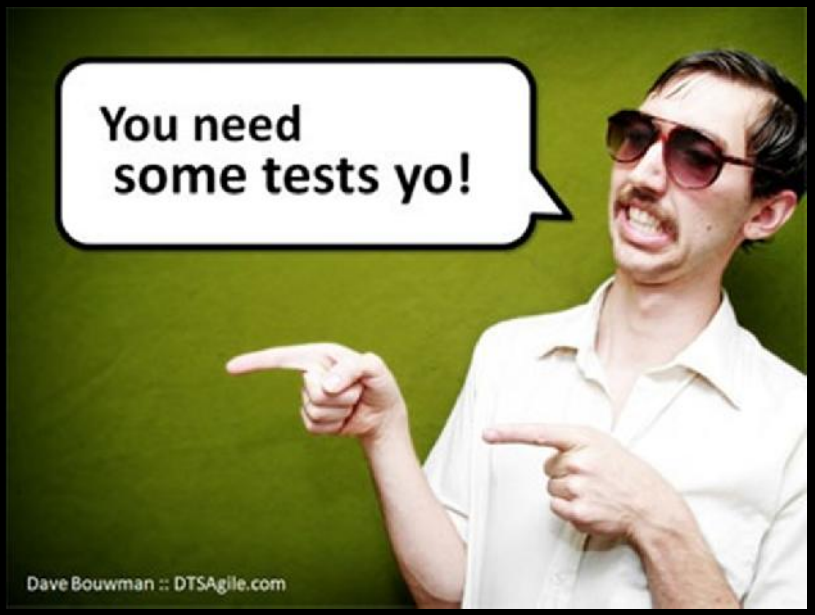
- Unit testing focuses on verification effort on the smallest unit of software design (the software component or module).
- Using the component level design description as a guide, important control paths are tested to uncover errors within the boundary of the module.
- The unit test is white box oriented and the step can be conducted in parallel for multiple components.



Unit Testing Procedure







Code Coverage

- Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested.
- It is a form of testing that looks at the code directly and as such comes under the heading of white box testing.

...js | Code Coverage Report for Depot Application ×

Total Coverage: 65.5 %

Filename	Coverage	Total	Not E...
test/integration/user_stories_test.rb	40.6 %	96	57
test/integration/dsl_user_stories_test.rb	48.6 %	148	76
app/controllers/application.rb	81.6 %	38	7
app/models/product.rb	89.4 %	66	7
app/helpers/application_helper.rb	100.0 %	29	0
test/test_helper.rb	100.0 %	38	0
fcntl	100.0 %	11	0
Total	65.5 %	426	147

Clear Results | Run All Tests | Done

Code Coverage Types

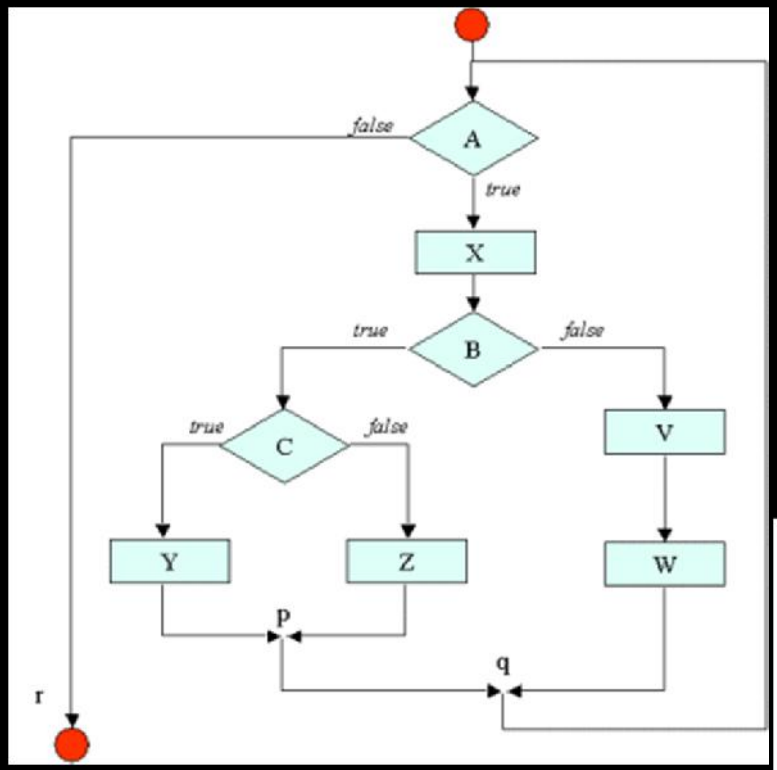
- Statement Coverage: In this type of testing the code is executed in such a manner that every statement of the application is executed at least once. It helps in assuring that all the statements execute without any side effect.
- Segment coverage: Each segment of code executed at least once.
- Branch Coverage: A branch is the outcome of a decision, so branch coverage simply measures which decision outcomes have been tested. Each branch in the code is taken in each possible direction at least once.
- Compound Condition Coverage: When there are multiple conditions, you must test not only each direction but also each possible combinations of conditions, which is usually done by using a „Truth Table".

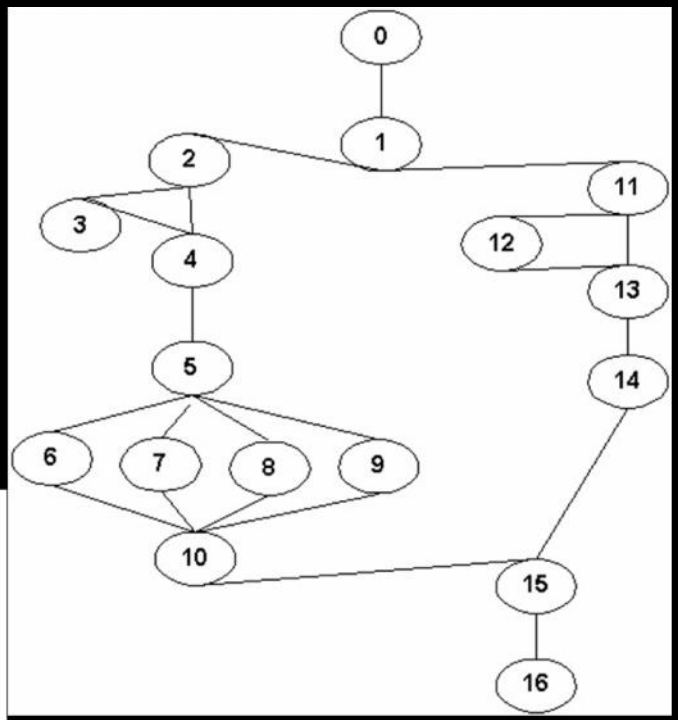
Code Coverage Types

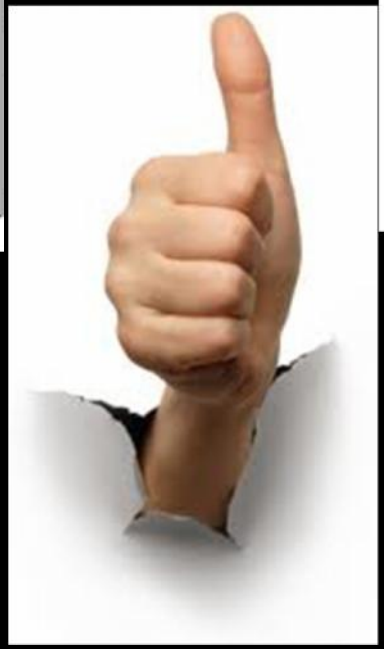
- Basis Path Testing: Each independent path through the code is taken in a pre-determined order.
- Data Flow Testing (DFT): In this approach you track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code.
- Loop Testing: This strategy relate to testing single loops, concatenated loops, and nested loops. Loops are fairly simple to test unless dependencies exist among the loop or b/w a loop and the code it contains.

Cyclomatic complexity

- Cyclomatic complexity is a software metric that provides a quantitative measure of the global complexity of a program.
- When this metric is used in the context of the basis path testing, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program.
- Cyclomatic complexity measures the amount of decision logic in a single software module. It gives the number of recommended test for software.
  Cyclomatic complexity is based entirely on the structure of software's control flow graph.



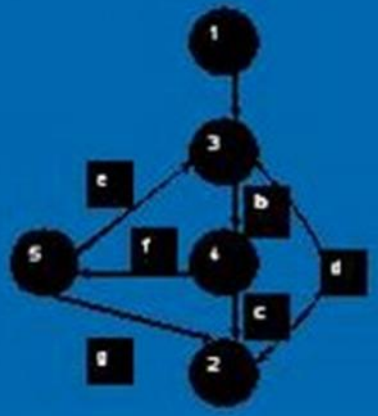




Graph Matrix

- Graph Matrix is a data structure that assists in basis path testing.
- A graph matrix is a square matrix whose size is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries corresponds to connections between nodes.
- Referring to the figure each node on the flow graph is identified by numbers, while each edge is identified by the letters. A letter entry is made in the matrix to correspond to a connection between two nodes. For example node 3 is connected to node 4 by edge b.

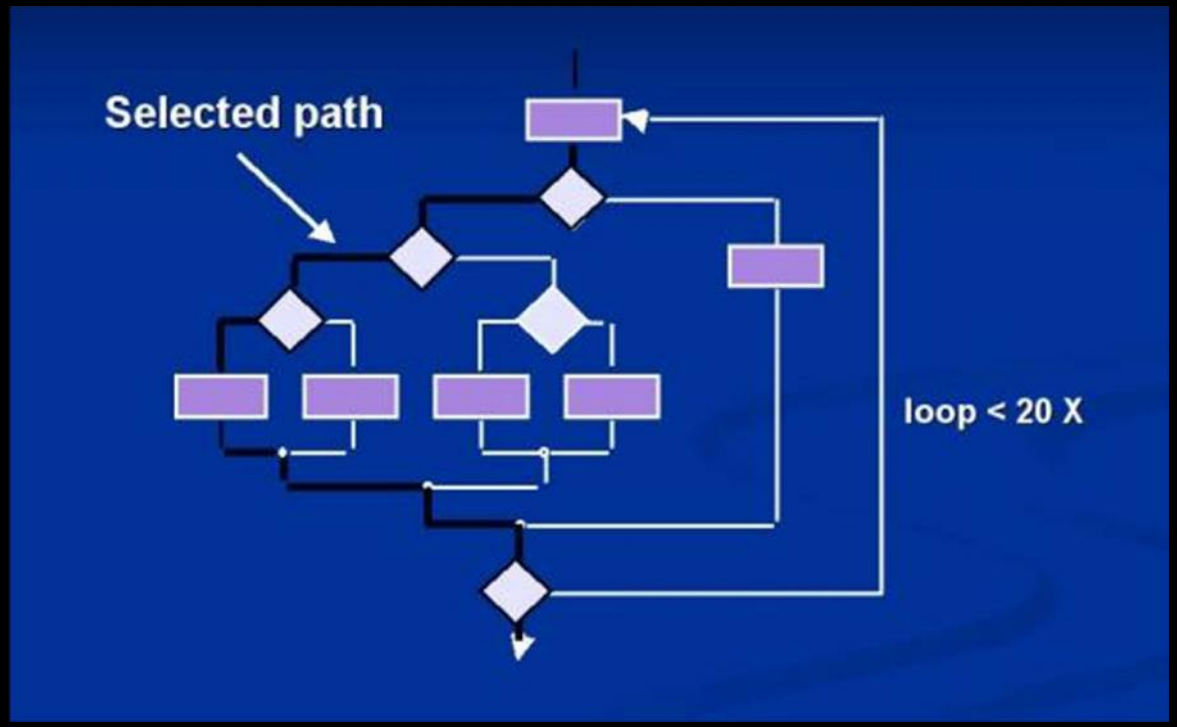
Graph Matrices



	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

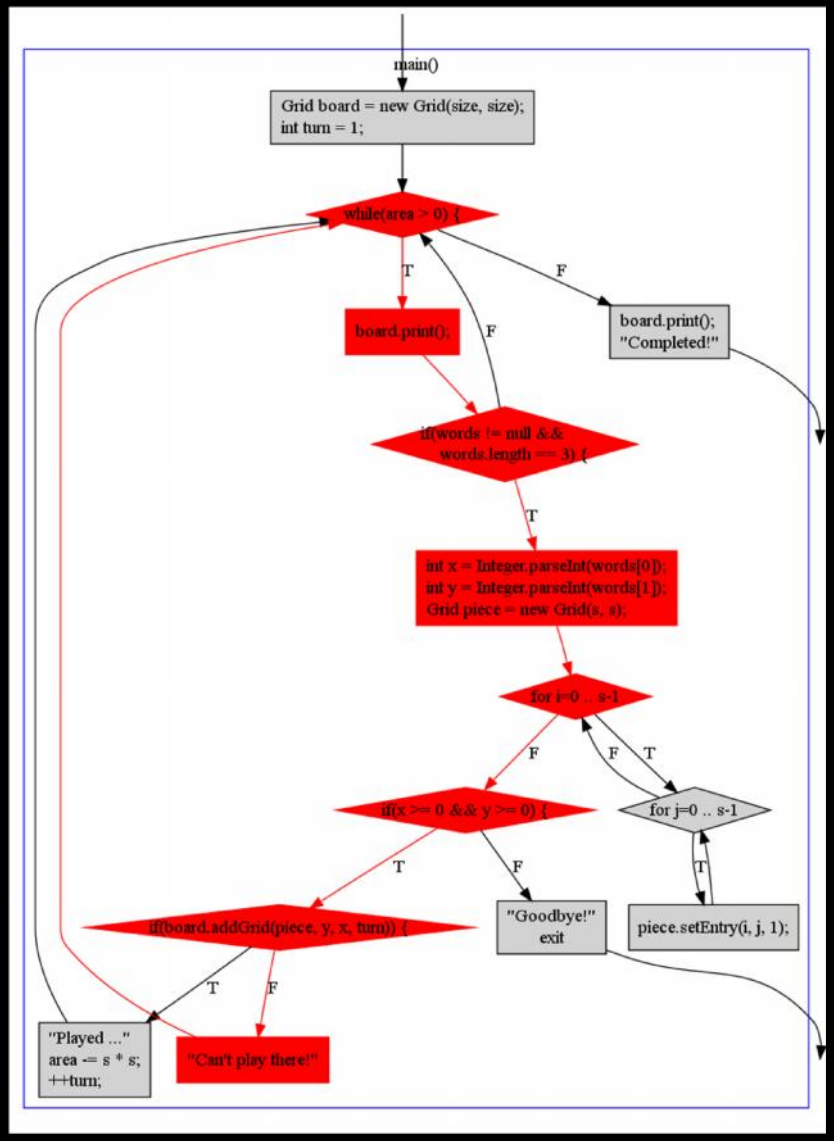
Basis Path Testing

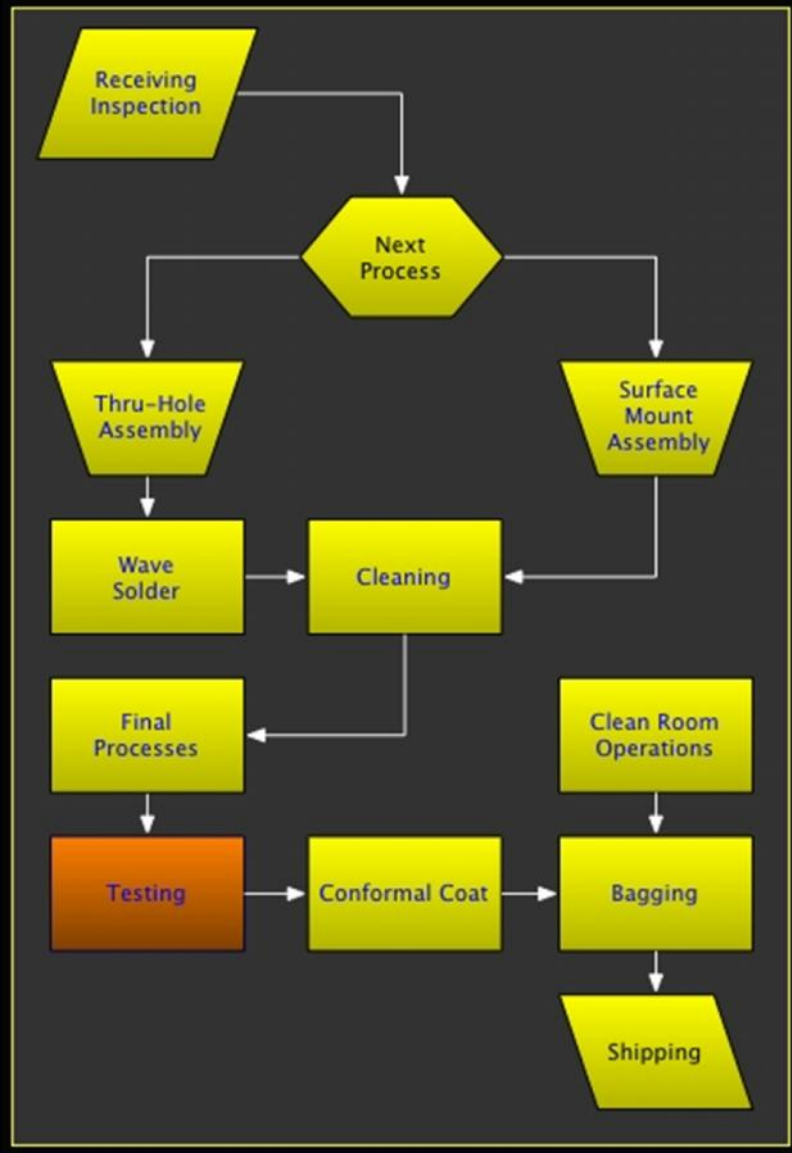
- Basis path testing (a white-box testing technique): - First proposed by Tom McCabe. - Used as a guide for defining a basis set of execution path. - Guarantee to execute every statement in the program at least one time. Step 1 : Using the design or code as a foundation, draw a corresponding flow graph. Step 2: Determine the cyclomatic complexity of the resultant flow graph. Step 3: Determine a basis set of linearly independent paths. For example, path 1: 1-2-4-5-6-7 path 2: 1-2-4-7 path 3: 1-2-3-2-4-5-6-7 path 4: 1-2-4-5-6-5-6-7 Step 4: Prepare test cases that will force execution of each path in the basis set. Step 5: Run the test cases and check their results



Control-Flow Testing

- Control-flow testing is a structural testing strategy that uses the program's control flow as a model.
- Control-flow testing techniques are based on judiciously selecting a set of test paths through the program.
- The set of paths chosen is used to achieve a certain measure of testing thoroughness.





Condition testing

- Condition testing is a test case design method that exercises the logical conditions contained in a program module. A simple condition is a Boolean variable or a relational expression. A relational expression takes the form, E1 < relational operator> E2 Where E1 and E2 are arithmetic expressions and < relational operator> is one of the following: {<, ?, >, ?, =, ?}
- A compound condition is composed of two or more simple conditions, Boolean operators and parenthesis.
- A condition without relational expression is referred to as a Boolean expression
- Types of error in a condition includes the following:
- Boolean operator error
- Boolean variable error
- Relational operator error
- Arithmetic expression error

```
from query in linqDataSource where query.Length > 5 select query;
```

```
linqDataSource.Where(query => query.Length > 5).Select(query => query);
```

Operator + Operand = Expression

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

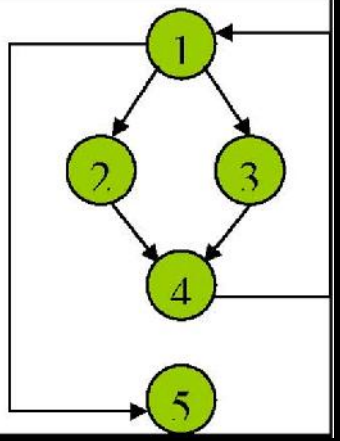
Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

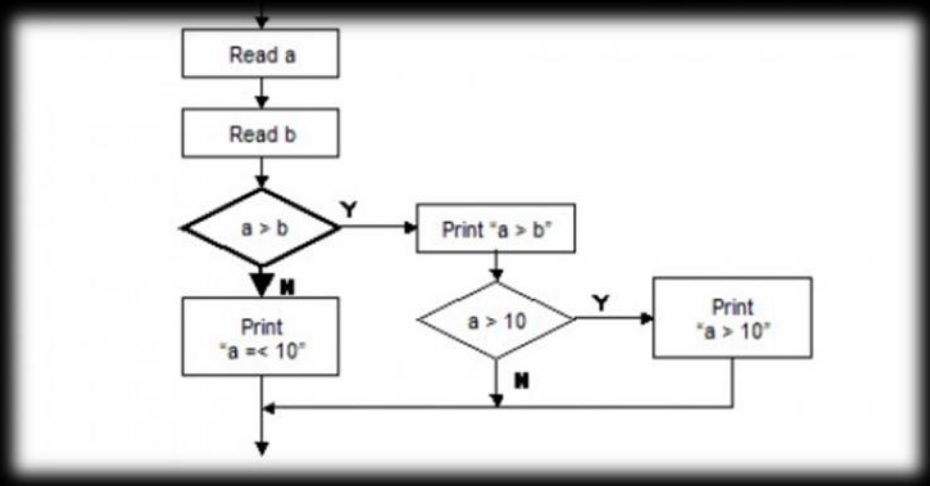
Branch Testing

- Branch testing is the simplest condition testing strategy.
- For a compound condition Com, the true and false branches of Com and every simple condition in Com need to be executed at least once.

statement and see how many test cases can possibly be developed.

```
for (i = 0; i < N; i++) {          //1
  if (condition1)
        // do something here      //2
  else
        // do something here      //3
  // something here               //4
}
                                   //5
```





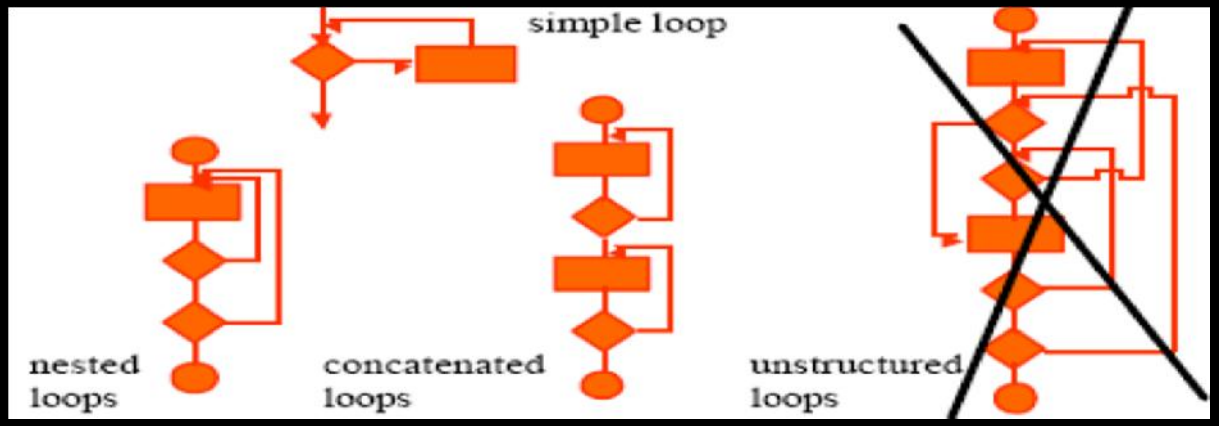
Domain testing

- Domain testing is an important white box testing method. The goal is to check values taken by a variable, a condition, or an index, and to prove whether they are outside the valid range or not. It also contains checking that the program accepts only valid input , because it is unlikely to get reasonable results if idiocy has been entered. This part can be called ``garbage in -- garbage out'' testing.

Loop testing

Loop testing is a white box testing technique that focuses on the validity of loop constructs.

- ✓ Simple loops
- ✓ Concatenated loops
- ✓ Nested loops



Loop Testing

The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.

1.Skip the loop entirely

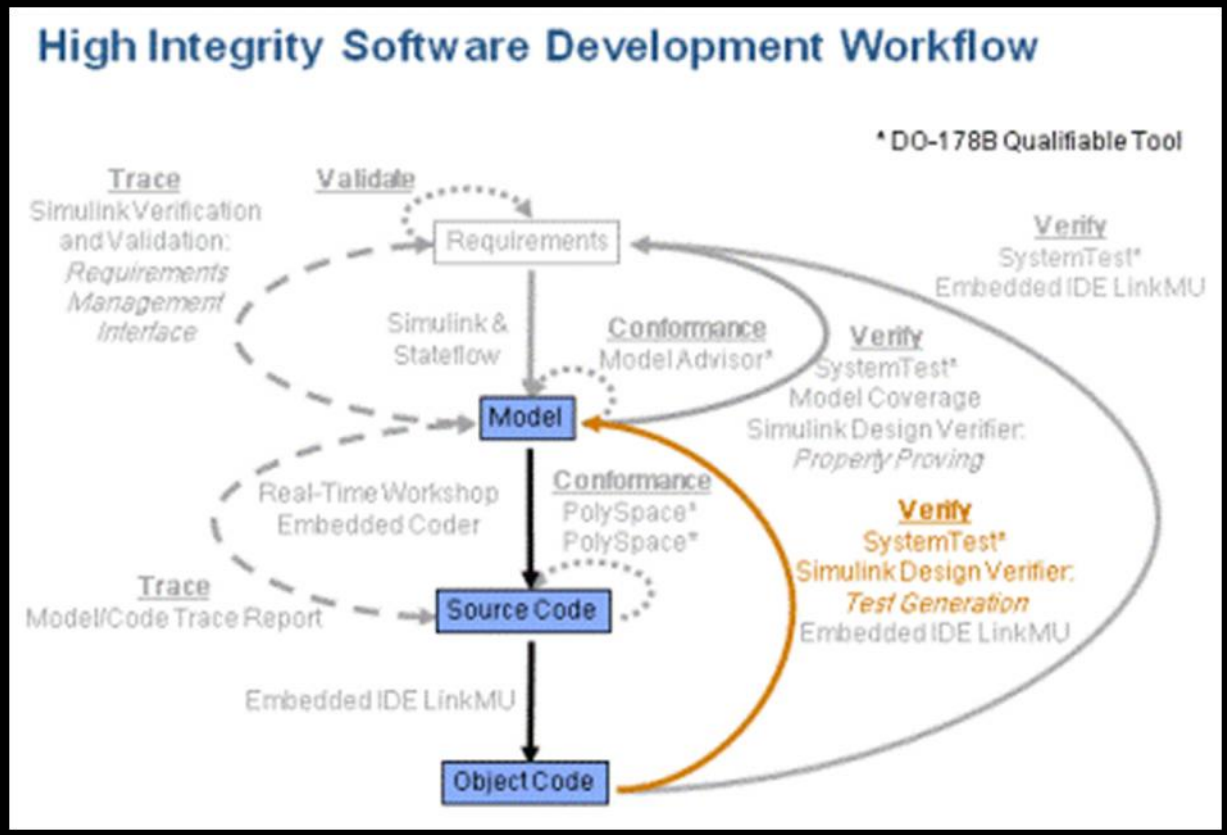
2.Only one pass through the loop

3.two passes through the loop

4.m passes through the loop where m<n

5.n-1,n,n+1 passes through the loop.

Loop Testing

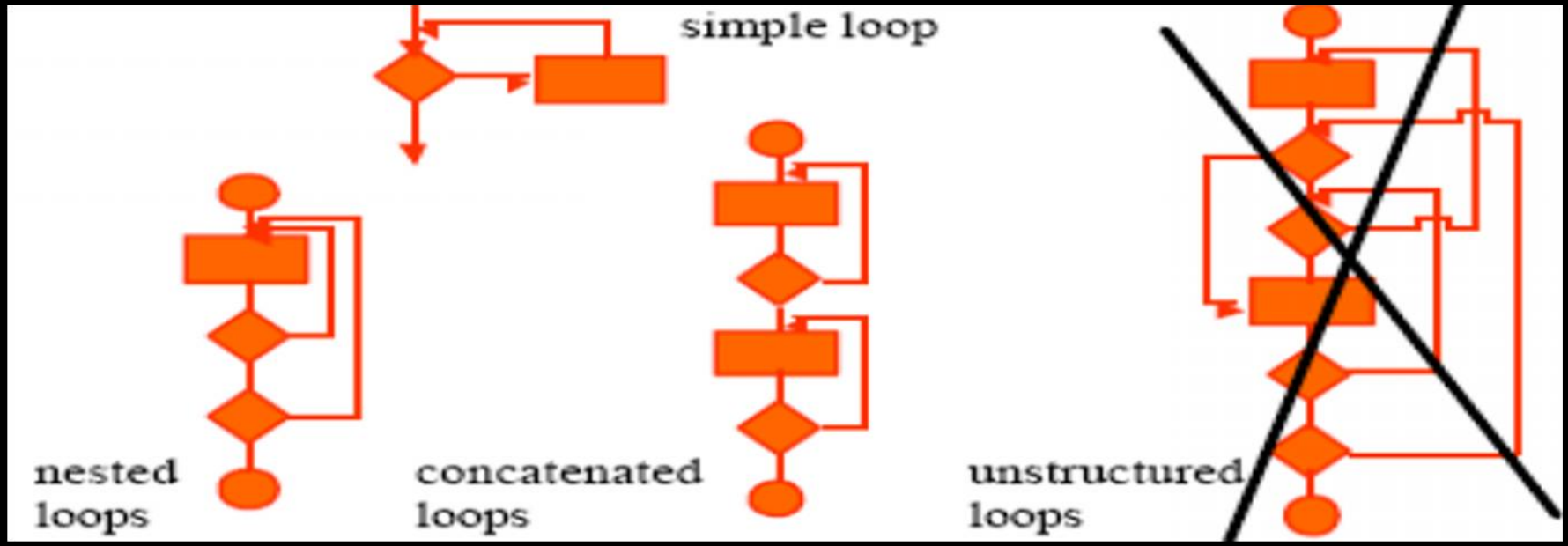


Nested Loops

If we were to extend the test approach for simple loops to nested loops the number of possible tests would grow as the level of nesting increases. Beizer suggest an approach that will help to reduce the number of tests:

1. **Start at the innermost loop. Set all other loops to min. values**
2. **Conduct simple loop tests for the innermost loop while holding the outer loops at their min. iteration parameter values.**
3. **Work outward conducting tests for the next loop, but keeping all other outer loops at min. values and other nested loops to typical values.**
4. **Continue until all loops have been tested.**

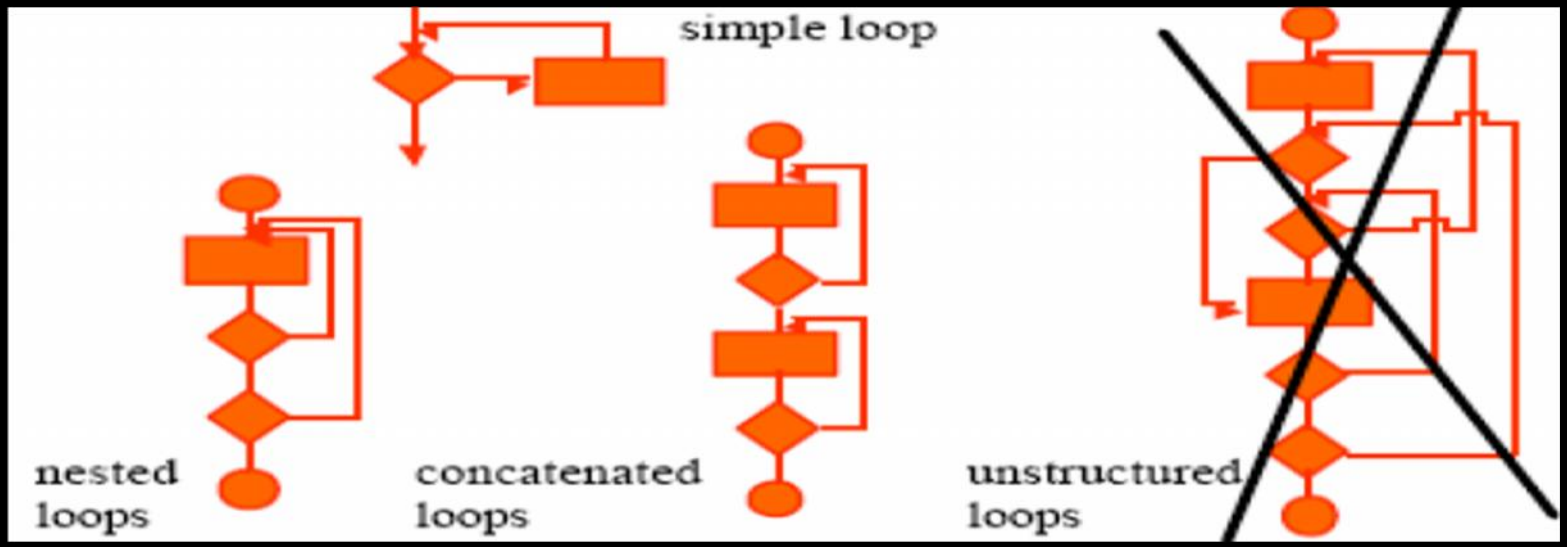
Nested Loops



Concatenated loops

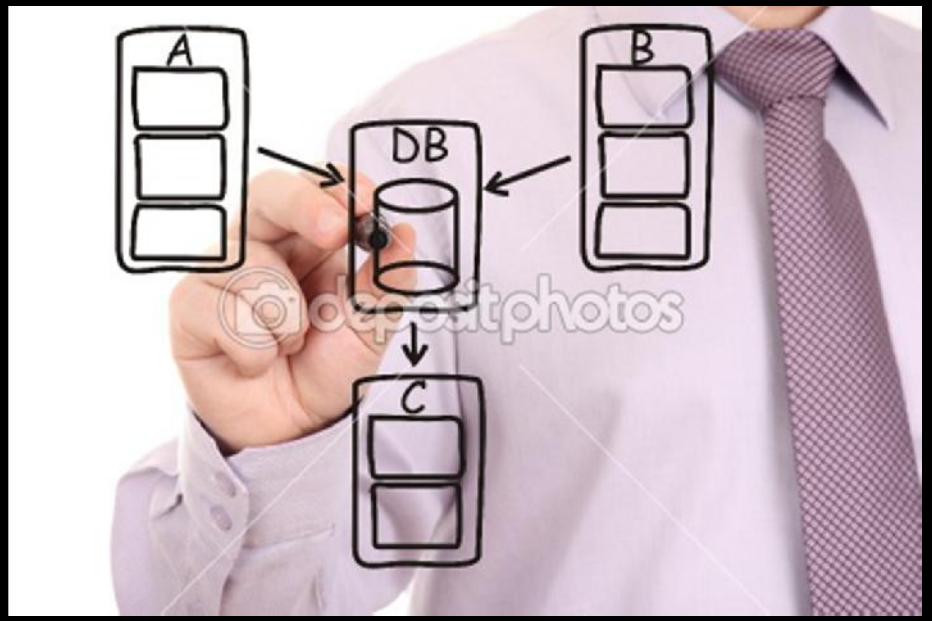
- Concatenated loops can be tested using the approach defined for simple loops, if each of the loops are independent of the other. However if two loops are concatenated and the loop counter for loop 1 is used as a initial value for loop 2, then the loops are not independent.
- When the loops are not independent, the approach applied to nested loops is recommended.

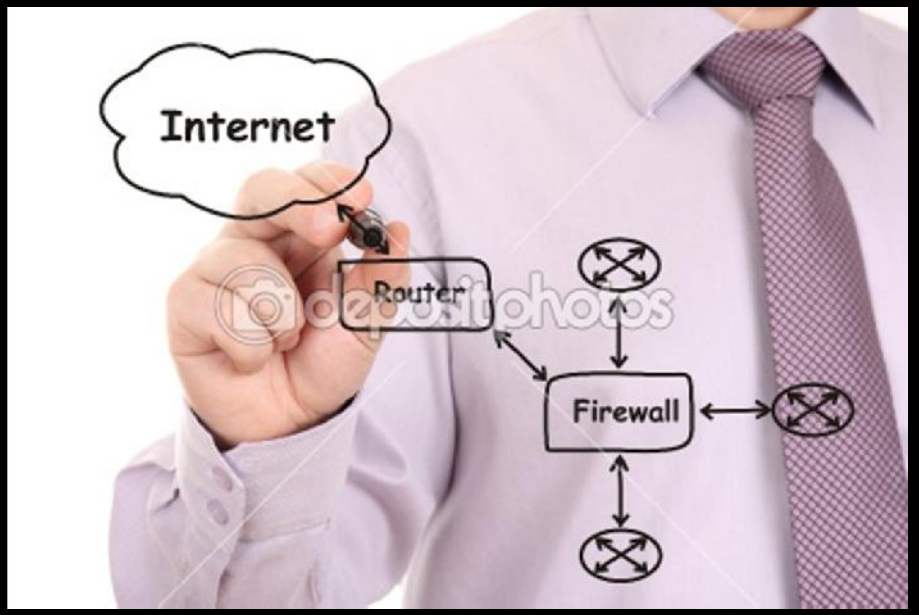
Concatenated loops



Data flow testing

- Data flow testing is a structural test technique which aims to execute subpaths from points where each variable in a component is defined to points where it is referenced.
- These subpaths are known as definition-use pairs (du-pairs). The different data flow coverage criteria require different du-pairs and subpaths to be executed.
- Data-flow testing is a white box testing technique that can be used to detect improper use of data values due to coding errors. Errors are inadvertently introduced in a program by programmers. For instance, a software programmer might use a variable without defining it.
- Additionally, he/she may define a variable, but not initialize it and then use that variable in a predicate.





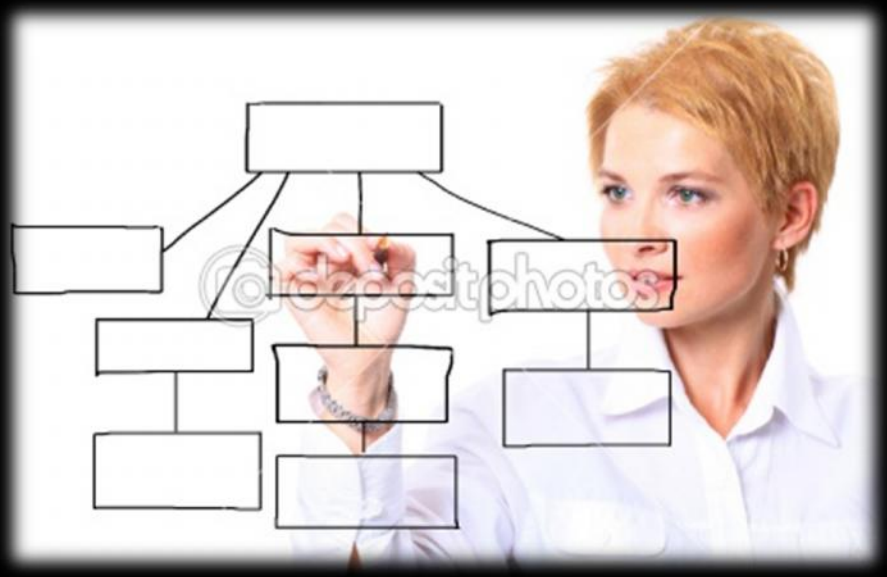
http://depositphotos.com/5232434/stock-photo-Woman-hand-drawing-chart-on-whiteboard-2.html

Data Flow Testing

There are four ways data can be used:

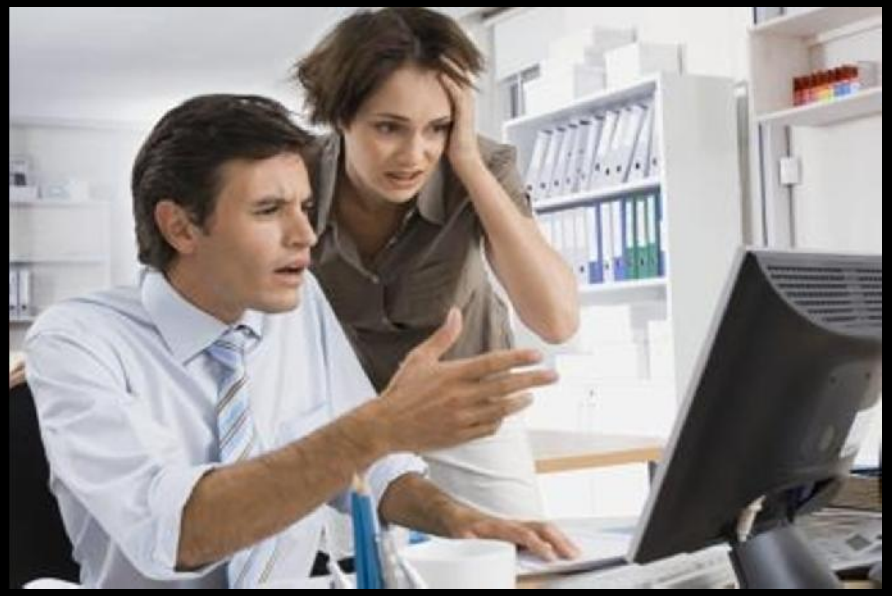
- ✓ defined,
- ✓ used in a predicate,
- ✓ used in a calculation, and
- ✓ killed.

Data-flow testing monitors the lifecycle of a piece of data and looks out for inappropriate usage of data during definition, use in predicates, computations and termination (killing). It identifies potential bugs by examining the patterns in which that piece of data is used.

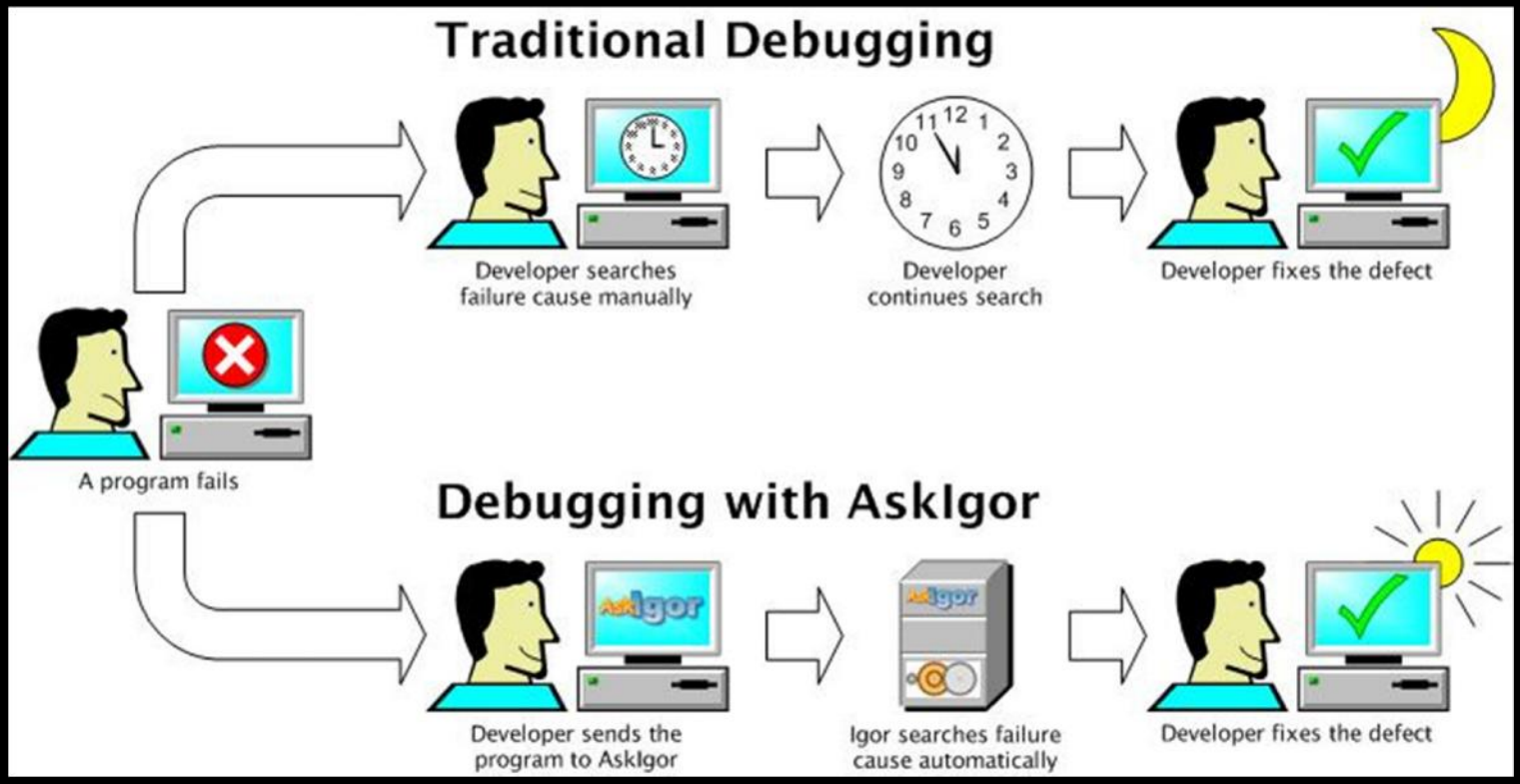


Debugging

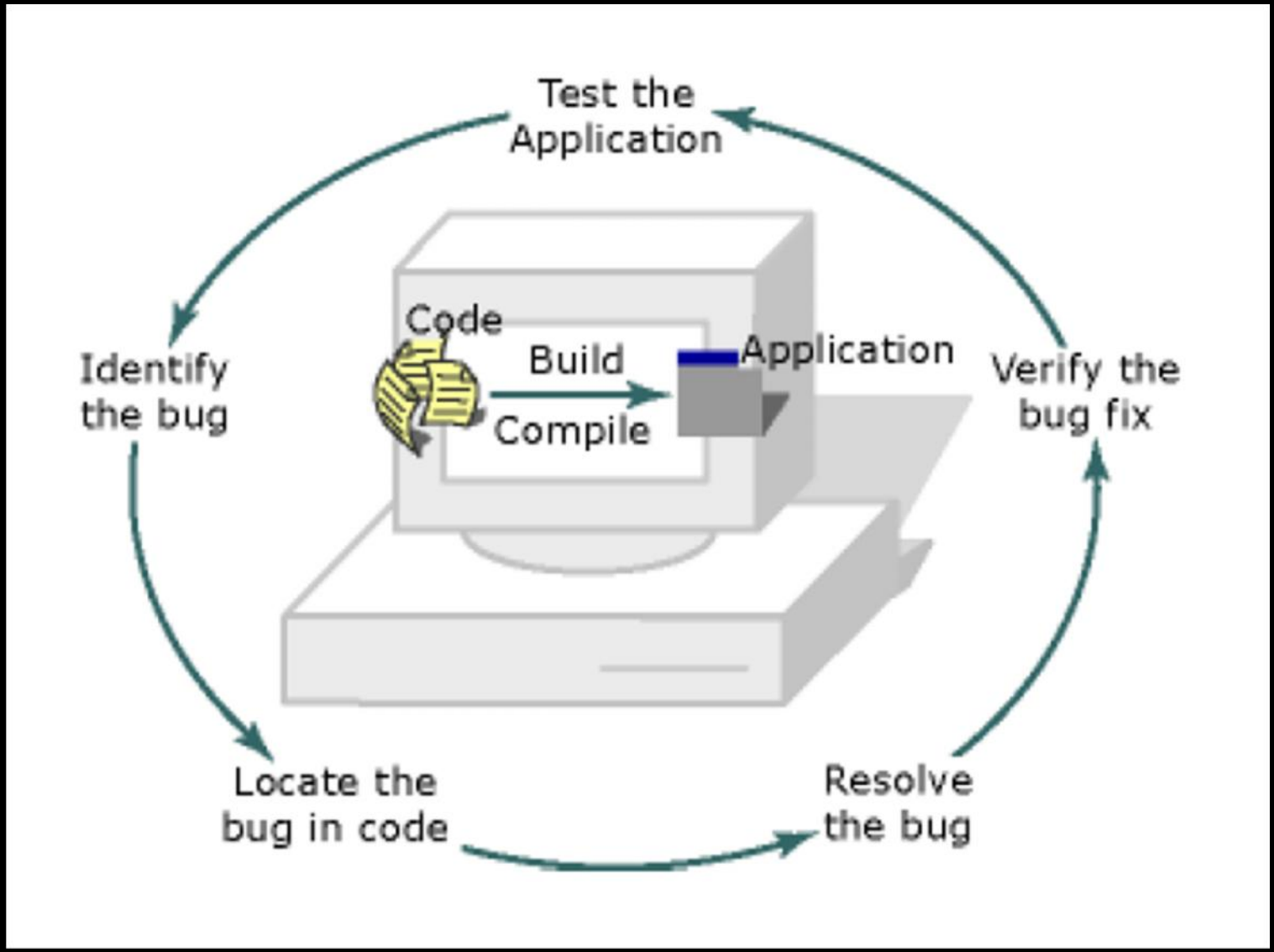
- Debugging is that activity which is performed after executing a successful test case.
- Remember that a successful test case is one that shows that a program does not do what it was designed to do**.**
- Debugging is a two-step process that begins when you find an error as a result of a successful test case.
- **Step 1 is the determination of the exact nature and location of the suspected error within the program.**
- **Step 2 consists of fixing the error.**
- *Locating the error represents about 95% of the activity*
- Process of finding the location of an error, given a suspicion that an error exists, based on the results of a successful test case.



Debugging



Debugging Lifecycle



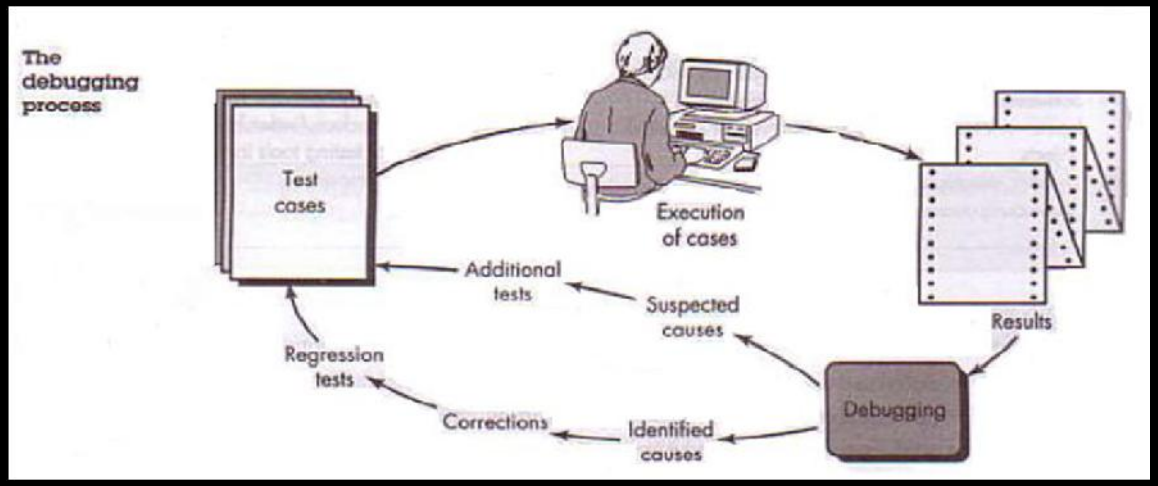
Debugging Process

Debugging will always have one of two outcomes:

1.The cause will be found and corrected or

2.The cause will not be found.

In the latter case, the person performing debugging may suspect a cause, design one or more test cases to help validate that suspicion, and work toward error correction in an iterative fashion.



Bug

What is a BUG

A fault in a program, which causes the program to perform in an unintended or unanticipated manner. Reports detailing bugs in a program are commonly known as **bug reports, fault reports, problem reports, trouble reports, defect reports etc. Why BUG Occurs**

There are so many reasons that can cause a bug some of them can be:

- ✓ **Syntax errors in the Codes**
- ✓ **logical errors in the Codes**
- ✓ **Unfinished requirements**
- ✓ **Misunderstanding of user needs**
- ✓ **Errors in the design documents**
- ✓ **Lack of documentation**

Bug life cycle

In software development process, the bug has a life cycle. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. **States of a bug:**

1) New
2) Open
3) Assign
4) Test
5) Verified
6) Deferred
7) Reopened
8) Rejected
9) Closed

Bug life cycle

Description

- New: When the bug is posted for the first time, its state will be NEW. This means that the bug is not yet approved.
- Open: After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as OPEN.
- Assign: Once the lead changes the state as OPEN, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.
- Resolved/Fixed/Test: When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed‟ and the bug is passed to testing team.
- Deferred: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

Description

- Rejected/Invalid: Some times developer or team lead can mark the bug as Rejected or invalid if the system is working according to specifications and bug is just due to some misinterpretation.
- Pending Retest: After the bug is fixed, it is passed back to the testing team to get retested and the status of „Pending Retest‟ is assigned to it.
- Retest: The testing team leader changes the status of the bug, which is previously marked with „Pending Retest‟ to „Retest‟ and assigns it to a tester for retesting.
- Duplicate: If the bug is repeated twice, then one bug status is changed to “DUPLICATE”.
- Verified: Once the bug is fixed and the status is changed to TEST, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to VERIFIED

Description

- Could not reproduce: If developer is not able to reproduce the bug by the steps given in bug report by QA then developer can mark the bug as „CNR". QA needs action to check if bug is reproduced and can assign to developer with detailed reproducing steps.
- Reopened: If the bug still exists even after the bug is fixed by the developer, the tester changes the status to REOPENED. The bug traverses the life cycle once again.
- Closed: Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to CLOSED. This state means that the bug is fixed, tested and approved.
- Postponed: Sometimes, testing of a particular bug has to be postponed for an indefinite period. This situation may occur because of many reasons, such as unavailability of Test data, unavailability of particular functionality etc. That time, the bug is marked with „Postponed" status

Common Types Of Bugs

Math's bugs

Division by zero

Arithmetic overflow or underflow

Logic bugs

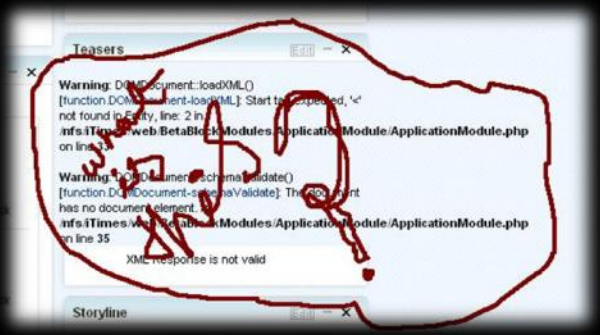
Infinite loops and infinite recursion

Syntax bugs

Use of the wrong operator, such as performing assignment instead of equality

Math's bugs

Logic bugs



Syntax bugs

```
{
    if (g_bModActive)
    {
        new team = GetClientTeam(client);

        if (team == Team_Zombies && !g_ClientInfo[client][Info_Equip])
        {
            return Plugin_Continue;
        }

        decl String:className[64];
        GetEdictClassname(weapon, className, sizeof(className));

        if (ReplaceString(className, sizeof(className), "weapon_", NULL_STRING)
        {
            switch (team)
```

Common Types Of Bugs

Resource bugs

Using an un-initialized variable

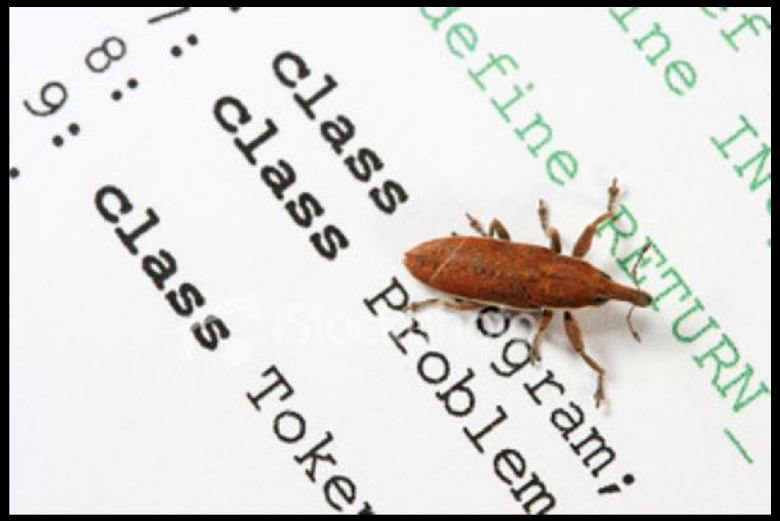
Resource leaks, where a finite system resource such as memory or file handles are exhausted by repeated allocation without release.

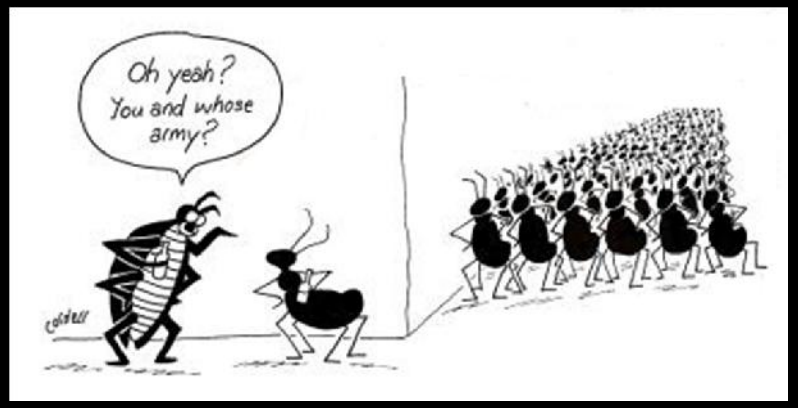
Buffer overflow, in which a program tries to store data past the end of allocated storage.

Team working bugs

Comments out of date or incorrect: many programmers assume the comments accurately describe the code

Differences between documentation and the actual product





Reporting

Daily Summary Data : In addition to entering new bugs and closing fixed bug in the bug tracking application,

Number of test cases completed

Number of new issues identified and reported

Number of test cases, which failed

The current list of existing bugs by severity

Weekly Summary Report : At the end of each week, creates a weekly report to provide some additional information to clients.

End of Cycle Reports: QA engineers execute all test cases for a build, supply a complete set of test results for the cycle. provide the following information:

List of bugs found in this testing cycle

List of bugs fixed in this testing cycle

List of bugs deferred to a later release

Types of Bugs/Defects

Defects that are detected by the tester are classified into categories by the nature of the defect. The following are the classification

Showstopper (X): The impact of the defect is severe and the system cannot go into the production environment without resolving the defect since an interim solution may not be available.

Critical (C): The impact of the defect is severe, however an interim solution is available.

Non critical (N): All defects that are not in the X or C category are deemed to be in the N category. These are also the defects that could potentially be resolved via documentation and user training. These can be GUI defects.

Defect Report

Particulars that have to be filled by a tester are:

Defect Id: Number associated with a particular defect, and henceforth referred by its ID.

Date of execution: The date on which the test case which resulted in a defect was executed.

Severity: As explained, it can be Critical, Non-Critical and Showstopper

Module ID: Module in which the defect occurred

Status:New,Open,Assign,Test,Verified,Deferred,Reopened,Rejected,Close

Defect description: Description as to how the defect was found, the exact steps that should be taken to simulate the defect, other notes and attachments if any.

Test Case Reference No: The number of the test case which resulted in the defect

Defect Report

Owner: The name of the tester who executed the test case

Test case description: The instructions in the test cases for the step in which the error occurred

Expected Result: The expected result after the execution of the instructions in the test case descriptions

History of the defect: Normally taken care of the automated tool used for defect tracking and reporting.

Attachments: The screen shot showing the defect should be captured and attached

Responsibility. Identified team member of the development team for fixing the defect.

Effective Bug Reporting

1. Bug Description should be clearly identifiable
2. Bug should be reported after building a proper context
3. Steps should be clear with short and meaningful sentences
4. Give references to specifications
5. Assign severity and priority
6. Provide Screenshots

Debugging Methods

The methods of debugging are listed below.

1.Debugging by Brute Force Attack

2.Debugging by Induction

3.Debugging by Deduction

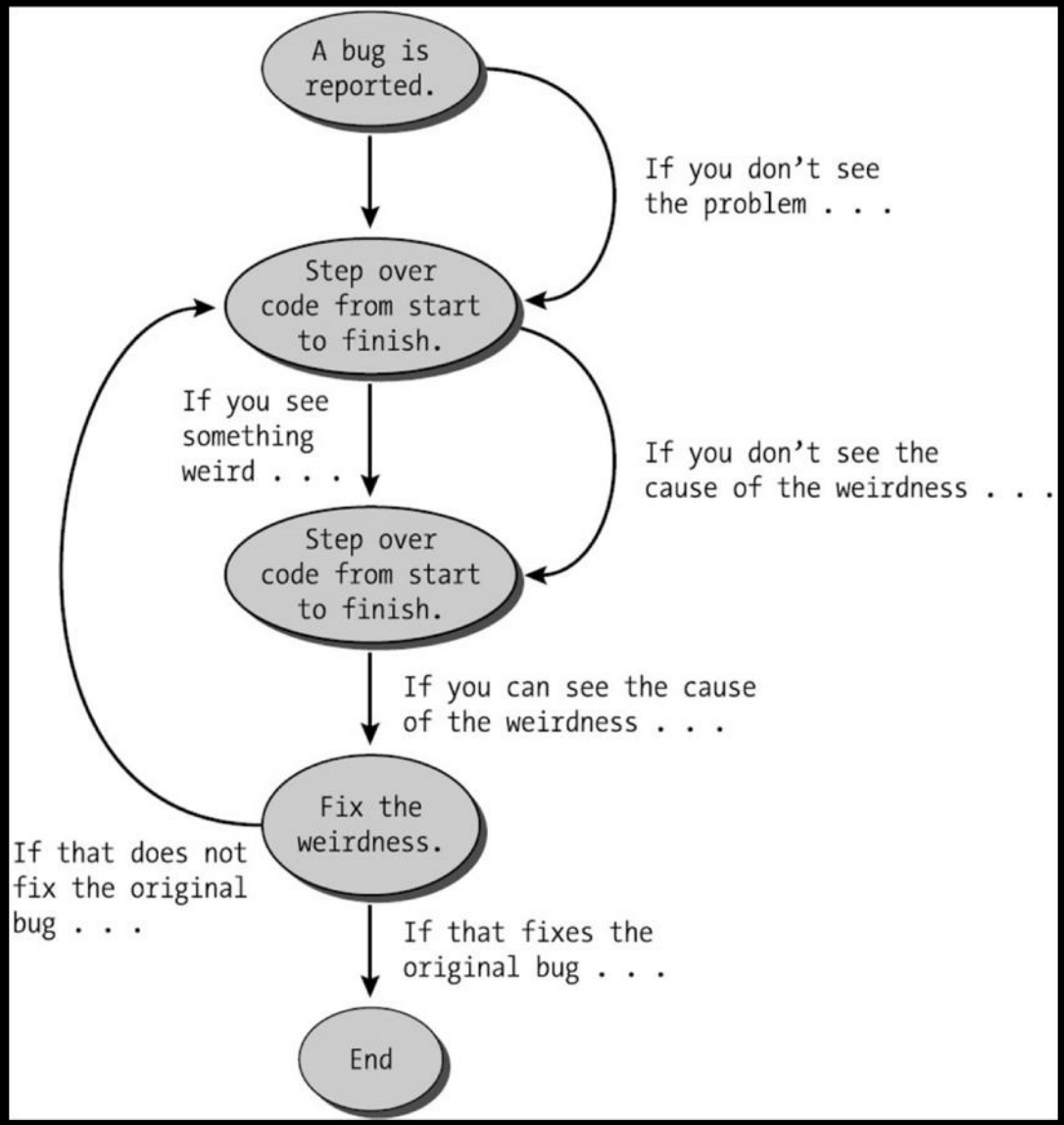
4.Debugging by Backtracking

5.Debugging by Testing

Debugging by Brute Force

- The most common scheme for debugging a program is the “brute force” method. It is popular because it requires little thought and is the least mentally taxing of the methods, but it is inefficient and generally unsuccessful.
- Brute force methods can be partitioned into at least three categories:

1. **Debugging with a storage dump.**
2. **Debugging according to the common suggestion to “scatter print statements throughout your program.”**
3. **Debugging with automated debugging tools.**



Brute force Method

Storage dump is the display or printout of the contents of memory.

When a program abends, a memory dump can be taken in order to examine the status of the program at the time of the crash.

The programmer looks into the buffers to see which data items were being worked on when it failed. Counters, variables, switches and flags are also inspected.

Brute force debugging method is applied when all else fails

It is the most inefficient of the brute force methods. Here‟s why;

There's a massive amount of data, most of which is irrelevant.

A memory dump is a static picture of the program, showing the state of the program at only one instant in time; to find errors, you have to study the dynamics of a program (state changes over time).

Brute force Method- Scattering Print Statements

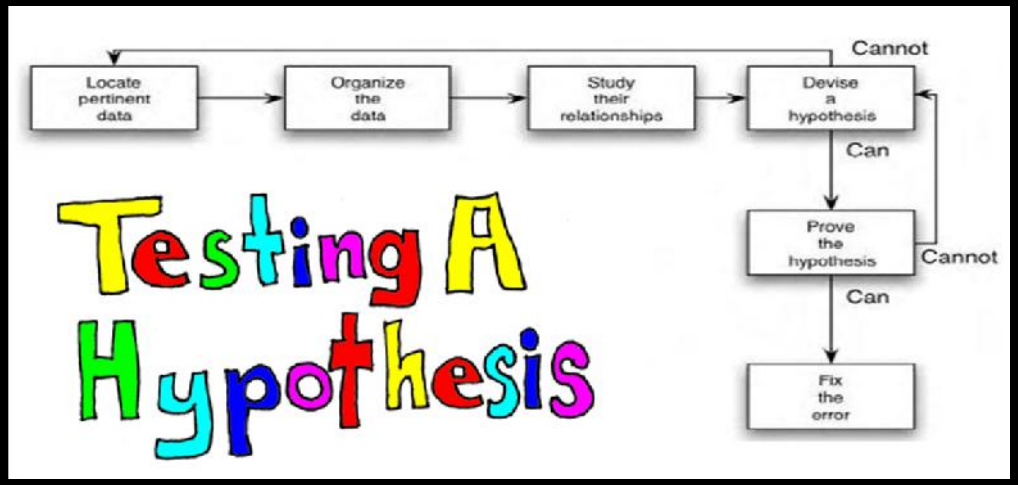
- Scattering statements throughout a failing program to display variable values is better than a dump as it is not static and shows the dynamics of a program, but this method, too, has many shortcomings:
- **Again you are not thinking.**
- **It requires you to change the program; such changes can mask the error, or introduce new errors.**
- **It may work on small programs, but the cost of using it in large programs like operating systems is quite large.**

Brute force Method- Automated Debugging Tools

- Automated debugging tools work similarly to inserting print statements within the program, but rather than making changes to the program, you analyze the dynamics of the program with the debugging features of the programming language.
- A common function of debugging tools is the ability to set breakpoints that cause the program to be suspended when a particular statement is executed or when a particular variable is altered, and then the programmer can examine the current state of the program.

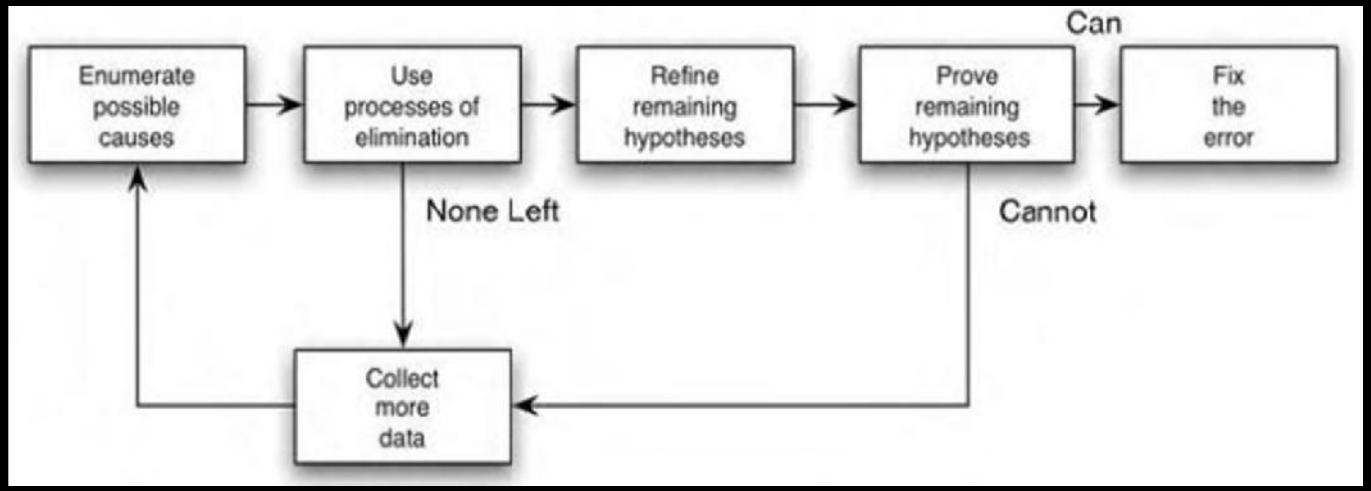
Induction

It should be obvious that careful thought will find most errors without the debugger even going near the computer. One particular thought process is induction, where you move from the particulars of a situation to the whole. That is, start with the clues (the symptoms of the error, possibly the results of one or more test cases) and look for relationships among the clues.



Deduction

The process of deduction proceeds from some general theories or premises, using the processes of elimination and refinement, to arrive at a conclusion (the location of the error).



Backtracking

- Backtracking is a fairly common debugging approach that can be used successfully in small programs
- Beginning at the site where a symptom has been uncovered, the source code is traced backwards till the error is found.
- Unfortunately as the no of source lines increases, the no of potential backward paths may become unmanageably large

Assignment 01

What Software Development methods are being used in developed Countries and why their projects failure rate is very low?

Due Date :

Questions?

If you have any query please feel free to ask

Phone: +92-51-9047-574

Fax: +92-51-9047-420

Email: fahad.khan@uettaxila.edu.pk

University Of Engineering & Technology Taxila Pakistan