Software Testing

The LNM Institute of
Information Technology

Course Instructor:
Saurabh Kumar
Assistant Professor,
CSE, LNMIIT, Jaipur

Semester: V
CSE: 0326 Software Engineering

October 18, 2020

Introduction

- Definition
    - IEEE: Testing means the process of analyzing a software item to detect the differences between existing and required condition (i.e. bugs) and to evaluate the feature of the software item.

Introduction

- Definition
  - IEEE: Testing means the process of analyzing a software item to detect the differences between existing and required condition (i.e. bugs) and to evaluate the feature of the software item.
  - Myers: Testing is the process of analyzing a program with the intent of finding an error.

Introduction

- Definition
  - IEEE: Testing means the process of analyzing a software item to detect the differences between existing and required condition (i.e. bugs) and to evaluate the feature of the software item.
  - Myers: Testing is the process of analyzing a program with the intent of finding an error.
- Primary objectives
  - Testing is a process of executing a program with the intent of finding an error.

Introduction

- Definition
  - IEEE: Testing means the process of analyzing a software item to detect the differences between existing and required condition (i.e. bugs) and to evaluate the feature of the software item.
  - Myers: Testing is the process of analyzing a program with the intent of finding an error.
- Primary objectives
  - Testing is a process of executing a program with the intent of finding an error.
  - A good test case is one that has a high probability of finding and as yet undiscovered error.

Introduction

- Definition
  - IEEE: Testing means the process of analyzing a software item to detect the differences between existing and required condition (i.e. bugs) and to evaluate the feature of the software item.
  - Myers: Testing is the process of analyzing a program with the intent of finding an error.
- Primary objectives
  - Testing is a process of executing a program with the intent of finding an error.
  - A good test case is one that has a high probability of finding and as yet undiscovered error.
  - A successful test is one that uncovers as yet undiscovered error.

Terminologies

- Error:
  - It refers to the discrepancy between computed, observed or measured value and the specified value.

Terminologies

- Error:
  - It refers to the discrepancy between computed, observed or measured value and the specified value.
  - Errors can be defined as the difference between actual output of software and correct output.

Terminologies

- Error:
  - It refers to the discrepancy between computed, observed or measured value and the specified value.
  - Errors can be defined as the difference between actual output of software and correct output.
- Fault: It is a condition that causes a system to fail in performing its required function.

Terminologies

- Error:
  - It refers to the discrepancy between computed, observed or measured value and the specified value.
  - Errors can be defined as the difference between actual output of software and correct output.
- Fault: It is a condition that causes a system to fail in performing its required function.
- Failure:
  - A software failure occurs if the behavior of the software is different from specified behavior.

Terminologies

- Error:
  - It refers to the discrepancy between computed, observed or measured value and the specified value.
  - Errors can be defined as the difference between actual output of software and correct output.
- Fault: It is a condition that causes a system to fail in performing its required function.
- Failure:
  - A software failure occurs if the behavior of the software is different from specified behavior.
  - It is a stage when system becomes unable to perform a required function according to the specification mentioned.

Software Testing

- Testing Principles
  - All tests should be traceable to customer requirements.
  - Tests should be planned long before testing begins.
  - The Pareto principle applies to software testing.
  - Testing should begin in the small and progress toward testing in the large.
  - Exhaustive testing is not possible.
  - To be most effective, testing should be conducted by an independent third party.

Software Testing

- Testing Principles
  - All tests should be traceable to customer requirements.
  - Tests should be planned long before testing begins.
  - The Pareto principle applies to software testing.
  - Testing should begin in the small and progress toward testing in the large.
  - Exhaustive testing is not possible.
  - To be most effective, testing should be conducted by an independent third party.
- Attributes of a Good Test
  - A good test has a high probability of finding an error.
  - A good test is not redundant.
  - A good test should be best of breed.
  - A good test should be neither too simple nor too complex.

White Box Testing

- Also known as Gloss Box Testing or Structural testing.
- The internal logic of software components is tested.
- It is a test case design method that uses the control structure of the procedural design test cases.
- It is done in the early stages of the software development.

White Box Testing

- Also known as Gloss Box Testing or Structural testing.
- The internal logic of software components is tested.
- It is a test case design method that uses the control structure of the procedural design test cases.
- It is done in the early stages of the software development.
- Using this testing technique, software engineers can derive test cases that
  - All independent paths within a module have been exercised at least once.
  - Exercised true and false both the paths of logical checking.
  - Execute all the loops within their boundaries.
  - Exercise internal data structures to ensure their validity.

White Box Testing

- Advantages:
  - As the knowledge of internal coding structure is prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively.
  - It helps in optimizing the code.
  - It helps in removing the extra lines of code, which can bring in hidden defects.
  - We can test the structural logic of the software.
  - Every statement is tested thoroughly.
  - Forces test developer to reason carefully about implementation.
  - Approximate the partitioning done by execution equivalence.
  - Reveals errors in hidden code.

White Box Testing

- Advantages:
  - As the knowledge of internal coding structure is prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively.
  - It helps in optimizing the code.
  - It helps in removing the extra lines of code, which can bring in hidden defects.
  - We can test the structural logic of the software.
  - Every statement is tested thoroughly.
  - Forces test developer to reason carefully about implementation.
  - Approximate the partitioning done by execution equivalence.
  - Reveals errors in hidden code.
- Disadvantages:
  - It does not ensure that the user requirements are fulfilled.
  - As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost.
  - It is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.
  - The tests may not be applicable in real world situation.
  - Cases omitted in the code could be missed out.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?
  - What class of input will make good test cases?

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?
  - What class of input will make good test cases?
  - Is the system particularly sensitive to certain input values?

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?
  - What class of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?
  - What class of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?
  - What data rates and data volume can the system tolerate?

Black Box Testing

- Also known as Closed Box Testing or Behavioral testing.
- It exercises the input and output domain of the program to uncover errors in program, function, behavior and performance.
- It is done in the later stage of the software development.
- It attempts to find errors related to
  - Missing functions or incorrect functions.
  - Errors created due to interfaces.
  - Errors in accessing external databases.
  - Performance related errors.
  - Behavior related errors.
  - Initialization and termination errors.
- The main focus is on the information domain. Test is designed for following questions:
  - How is functionality validation testing?
  - What class of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?
  - What data rates and data volume can the system tolerate?
  - What effects will specific combinations of data have on system operation?

Black Box Testing

- Advantages:
  - More effective on larger units of code than glass box testing.
  - Tester needs no knowledge of implementation, including specific programming languages.
  - Tester and programmer are independent of each other.
  - Tests are done from a user's point of view.
  - Will help to expose any ambiguities or inconsistencies in the specifications.
  - Test cases can be designed as soon as the specifications are complete.

Black Box Testing

- Advantages:
  - More effective on larger units of code than glass box testing.
  - Tester needs no knowledge of implementation, including specific programming languages.
  - Tester and programmer are independent of each other.
  - Tests are done from a user's point of view.
  - Will help to expose any ambiguities or inconsistencies in the specifications.
  - Test cases can be designed as soon as the specifications are complete.
- Disadvantages:
  - Only a small number of possible inputs can actually be tested.
  - Without clear and concise specifications, test cases are hard to design.
  - There may be unnecessary repetition of test inputs if the tester is not informed of test cases the programmer has already tried.
  - May leave many program paths untested.
  - Cannot be directed toward specific segments of code which may be very complex (and therefore more error prone).
  - Most testing related research has been directed toward glass box testing.

Basis Path Testing

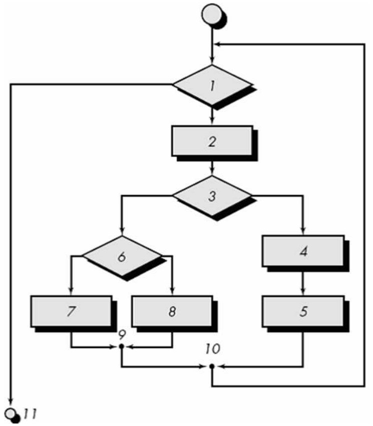

Figure 1: Flow Chart

Control Structure Testing

- Condition Testing
  - 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, possibly preceded with one NOT (¬) operator. A relational expression takes the form

    $$E_1 < relational - operator > E_2$$

    where, E_1 and E_2 are arithmetic expressions and ¡relational-operator¿ is one of the following: $<$, $\leq$, $=$, $\neq$ (nonequality), $>$, or $\geq$.
  - A compound condition is composed of two or more simple conditions, Boolean operators, and parentheses.
  - The Boolean operators allowed in a compound condition include OR (|), AND (&) and NOT (¬).
  - A condition without relational expressions is referred to as a Boolean expression.
  - The possible types of elements in a condition include a Boolean operator, a Boolean variable, a pair of Boolean parentheses (surrounding a simple or compound condition), a relational operator, or an arithmetic expression.
  - If a condition is incorrect, then at least one component of the condition is incorrect.

Control Structure Testing

- Condition Testing
  - Therefore, types of errors in a condition include the following:
    - Boolean operator error (incorrect/missing/extra Boolean operators)
    - Boolean variable error
    - Boolean parenthesis error
    - Relational operator error
    - Arithmetic expression error
  - The condition testing method focuses on testing each condition in the program.
  - Condition testing strategies generally have two advantages.
    - Measurement of test coverage of a condition is simple.
    - The test coverage of conditions in a program provides guidance for the generation of additional tests for the program.
  - The purpose of condition testing is to detect not only errors in the conditions of a program but also other errors in the program.
  - If a test set for a program P is effective for detecting errors in the conditions contained in P, it is likely that this test set is also effective for detecting other errors in P.
  - In addition, if a testing strategy is effective for detecting errors in a condition, then it is likely that this strategy will also be effective for detecting errors in a program.

Control Structure Testing

- Data Flow Testing
  - It selects test paths of a program according to the locations of definitions and uses of variables in the program.
  - Assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables.
  - For a statement with S as its statement number, let
    - DEF(S) = X | statement S contains a definition of X
    - USE(S) = X | statement S contains a use of X
  - If statement S is an *if* or *loop* statement, its DEF set is empty and its USE set is based on the condition of statement S.
  - The definition of variable X at statement S is said to be live at statement S' if there exists a path from statement S to statement S' that contains no other definition of X.
  - A definition-use (DU) chain of variable X is of the form $[X, S, S']$, where S and S' are statement numbers, X is in $DEF(S)$ and $USE(S')$, and the definition of X in statement S is live at statement S'.
  - One simple data flow testing strategy is to require that every DU chain be covered at least once.
  - This strategy is known as the *DU testing strategy*.

Control Structure Testing

- Data Flow Testing
  - It has been shown that DU testing does not guarantee the coverage of all branches of a program.
  - However, a branch is not guaranteed to be covered by DU testing only in rare situations such as *if-then-else* constructs in which
    - *then* part has no definition of any variable, and
    - *else* part does not exist.
  - In this situation, the *else* branch of the *if* statement is not necessarily covered by DU testing.
  - Data flow testing strategies are useful for selecting test paths of a program containing *nested if* and *loop* statements.

Control Structure Testing

- Loop Testing
  - Loop testing is a white-box testing technique that focuses exclusively on the validity of loop constructs.
  - Four different classes of loops can be defined: *simple loops*, *concatenated loops*, *nested loops*, and *unstructured loops*.
  - **Simple Loops:** following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.
    - Skip the loop entirely.
    - Only one pass through the loop.
    - Two passes through the loop.
    - m passes through the loop where m ¡ n.
    - n–1, n, $n + 1$ passes through the loop.

Control Structure Testing

- 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 geometrically as the level of nesting increases. This would result in an impractical number of tests. Beizer suggests an approach that will help to reduce the number of tests:
    - Start at the innermost loop. Set all other loops to minimum values.
    - Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
    - Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to “typical” values.
    - Continue until all loops have been tested.
  - **Concatenated loops:**
    - These can be tested using the approach defined for simple loops, if each of the loops is independent of the other.
    - However, if two loops are concatenated and the loop counter for loop 1 is used as the 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.
  - **Unstructured loops:** Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs.

Control Structure Testing

- Loop Testing

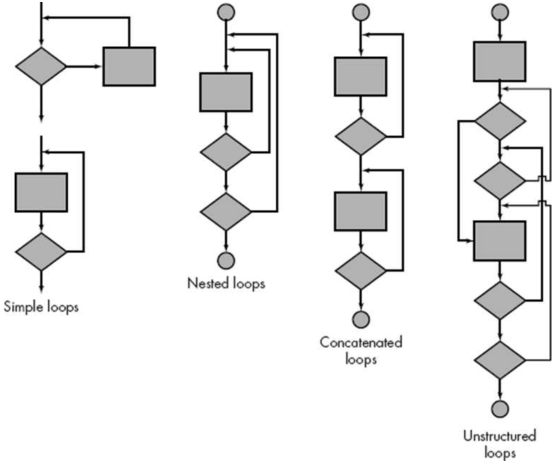

Figure 2: Classes of Loops

A Software Testing Strategy

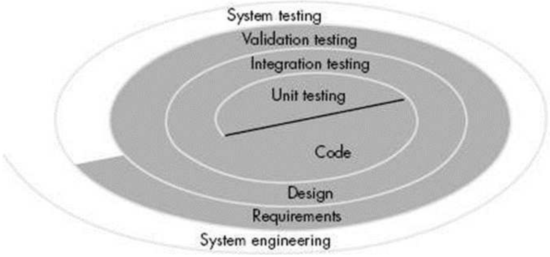

Figure 3: Software Testing

- Initially, system engineering defines the role of software and leads to software requirements analysis.
- Information domain, function, behavior, performance, constraints, and validation criteria for software are established during requirement analysis.
- To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.
  - Understand the users of the software and develop a profile for each user category.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.
  - Understand the users of the software and develop a profile for each user category.
  - Develop a testing plan that emphasizes “rapid cycle testing.”

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.
  - Understand the users of the software and develop a profile for each user category.
  - Develop a testing plan that emphasizes "rapid cycle testing."
  - Build "robust" software that is designed to test itself.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.
  - Understand the users of the software and develop a profile for each user category.
  - Develop a testing plan that emphasizes "rapid cycle testing."
  - Build "robust" software that is designed to test itself.
  - Use effective formal technical reviews as a filter prior to testing.

Various Levels of Testing

- Basic Levels of testing are
  - Unit Testing – Code
  - Validation Testing – Requirement
  - Integration Testing – Design
  - System Testing – System engineering
- Guidelines for successful testing strategy
  - Specify product requirements in a quantifiable manner long before testing commences.
  - State testing objectives explicitly.
  - Understand the users of the software and develop a profile for each user category.
  - Develop a testing plan that emphasizes “rapid cycle testing.”
  - Build “robust” software that is designed to test itself.
  - Use effective formal technical reviews as a filter prior to testing.
  - Conduct formal technical reviews to assess the test strategy and test cases themselves.

Various Levels of Testing

- Basic Levels of testing are
    - Unit Testing – Code
    - Validation Testing – Requirement
    - Integration Testing – Design
    - System Testing – System engineering
- Guidelines for successful testing strategy
    - Specify product requirements in a quantifiable manner long before testing commences.
    - State testing objectives explicitly.
    - Understand the users of the software and develop a profile for each user category.
    - Develop a testing plan that emphasizes "rapid cycle testing."
    - Build "robust" software that is designed to test itself.
    - Use effective formal technical reviews as a filter prior to testing.
    - Conduct formal technical reviews to assess the test strategy and test cases themselves.
    - Develop a continuous improvement approach for the testing process.

Unit Testing

- It is essentially for verification of the code produced during the code phase.
- The basic objective of this phase is to test the internal logic of the module.
- Unit testing makes heavy use of the white box testing technique.
- It exercises specific paths in a module control structure to ensure complete coverage and maximum error detection.
- In Unit testing, the module interface is tested to ensure that information properly flows into and out of the program unit.
- Also the data structure is tested to ensure that data stored temporarily maintain its integrity during execution or not.

Unit Testing

- Among the more common errors in computation are
  - Misunderstood or incorrect arithmetic precedence,
  - Mixed mode operations,
  - Incorrect initialization,
  - Precision inaccuracy,
  - Incorrect symbolic representation of an expression.

Unit Testing

- Among the more common errors in computation are
  - Misunderstood or incorrect arithmetic precedence,
  - Mixed mode operations,
  - Incorrect initialization,
  - Precision inaccuracy,
  - Incorrect symbolic representation of an expression.
- Test cases should uncover errors such as
  - Comparison of different data types,
  - Incorrect logical operators or precedence,
  - Expectation of equality when precision error makes equality unlikely,
  - Incorrect comparison of variables,
  - Improper or nonexistent loop termination,
  - Failure to exit when divergent iteration is encountered, and
  - Improperly modified loop variables.

Unit Testing

- Among the potential errors that should be tested when error handling is evaluated are
  - Error description is unintelligible.
  - Error noted does not correspond to error encountered.
  - Error condition causes system intervention prior to error handling.
  - Exception-condition processing is incorrect.
  - Error description does not provide enough information to assist in the location of the cause of the error.
- Boundary testing is the last (and probably most important) task of the unit test step.

Unit Testing

- Advantages
  - Can be applied directly to object code and does not require processing source code.
  - Performance profilers commonly implement this measure.

Unit Testing

- Advantages
  - Can be applied directly to object code and does not require processing source code.
  - Performance profilers commonly implement this measure.
- Disadvantages
  - Insensitive to some control structures (number of iterations)
  - Does not report whether loops reach their termination condition
  - Statement coverage is completely insensitive to the logical operators (OR and AND).

Integration Testing

- It uses the address of verification and program construction.
- Black box testing technique is widely used in this testing strategy.
- A limited amount of white box testing may be used to ensure coverage of control paths.
- The basic emphasis of this testing the interfaces between the modules.
- The objective is to take unit tested components and build a program structure that has been dictated by design.
- Types: top-down integration, bottom-up integration, regression, smoke.

Top-Down Integration

- It is an incremental approach to construction of program structure.
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).
- Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.

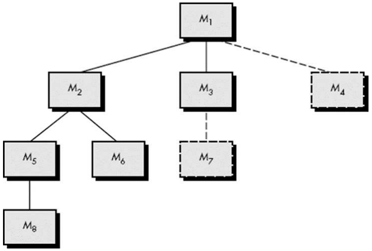

Figure 4: Top-Down Integration

Top-Down Integration

- Five steps of integration

Top-Down Integration

- Five steps of integration
    - The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

Top-Down Integration

- Five steps of integration
  - The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
  - Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

Top-Down Integration

- Five steps of integration
  - The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
  - Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
  - Tests are conducted as each component is integrated.

Top-Down Integration

- Five steps of integration
  - The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
  - Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
  - Tests are conducted as each component is integrated.
  - On completion of each set of tests, another stub is replaced with the real component.

Top-Down Integration

- Five steps of integration
  - The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
  - Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
  - Tests are conducted as each component is integrated.
  - On completion of each set of tests, another stub is replaced with the real component.
  - Regression testing may be conducted to ensure that new errors have not been introduced.

Bottom-Up Integration

- It begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure).
- Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

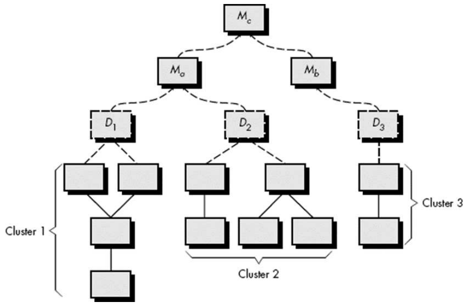

Figure 5: Bottom-Up Integration

Bottom-Up Integration

- A bottom-up integration strategy may be implemented with the following steps:

Bottom-Up Integration

- A bottom-up integration strategy may be implemented with the following steps:
  - Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.

Bottom-Up Integration

- A bottom-up integration strategy may be implemented with the following steps:
  - Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.
  - A driver (a control program for testing) is written to coordinate test case input and output.

Bottom-Up Integration

- A bottom-up integration strategy may be implemented with the following steps:
  - Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.
  - A driver (a control program for testing) is written to coordinate test case input and output.
  - The cluster is tested.

Bottom-Up Integration

- A bottom-up integration strategy may be implemented with the following steps:
  - Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.
  - A driver (a control program for testing) is written to coordinate test case input and output.
  - The cluster is tested.
  - Drivers are removed and clusters are combined moving upward in the program structure.

Regression Testing

- Each time a new module is added as part of integration testing, the software changes.
- New data flow paths are established, new I/O may occur, and new control logic is invoked.
- The regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.
- Capture/playback tools enable the software engineer to capture test cases and results for subsequent playback and comparison.
- The regression test suite contains three different classes of test cases:
  - A representative sample of tests that will exercise all software functions.
  - Additional tests that focus on software functions that are likely to be affected by the change.
  - Tests that focus on the software components that have been changed.

Smoke Testing

- It is an integration testing approach that is commonly used when "shrink – wrapped" software products are being developed.
- It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess its project on a frequent basis.
- The smoke testing approach encompasses the following activities:
  - Software components that have been translated into code are integrated into a "build." A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.
  - The build is integrated with other builds and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.

Smoke Testing

- Smoke testing provides a number of benefits when it is applied on complex, time critical software engineering projects:
  - **Integration risk is minimized:** Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered.
  - **The quality of the end-product is improved:** Because the approach is construction (integration) oriented, smoke testing is likely to uncover both functional errors and architectural and component-level design defects. If these defects are corrected early, better product quality will result.
  - **Error diagnosis and correction are simplified:** Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with "new software increments"—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.
  - **Progress is easier to assess:** With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

Validation Testing

- At the culmination of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected, and a final series of software tests, i.e. validation testing, may begin.
- Validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- At this point, a battle-hardened software developer might protest: "Who or what is the arbiter of reasonable expectations?"
- Reasonable expectations are defined in the Software Requirements Specification.
- The specification contains a section called Validation Criteria.
- Information contained in that section forms the basis for a validation testing approach.

Validation Testing

- Configuration Review:
  - An important element of the validation process is a configuration review.
  - The intent of the review is to ensure that all elements of the software configuration have been properly developed, are catalogued, and have the necessary detail to bolster the support phase of the software life cycle.
- Alpha Testing:
  - The alpha test is conducted at the developer's site by a customer.
  - The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.
- Beta Testing:
  - The beta test is conducted at one or more customer sites by the end-user of the software.
  - Unlike alpha testing, the developer is generally not present.
  - Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.
  - The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.
  - Software engineers make modifications and then prepare for release of the software product to the entire customer base.

System Testing

- Software is the only one element of a larger computer-based system.
- Ultimately, software is incorporated with other system elements (e.g., hardware, people, information), and a series of system integration and validation tests are conducted.
- These tests fall outside the scope of the software process and are not conducted solely by software engineers.
- However, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.
- A classic system testing problem is "finger-pointing."
- This occurs when an error is uncovered, and each system element developer blames the other for the problem.

System Testing

- Rather than indulging in such nonsense, the software engineer should anticipate potential interfacing problems and
  - Design error-handling paths that test all information coming from other elements of the system,
  - Conduct a series of tests that simulate bad data or other potential errors at the software interface,
  - Record the results of tests to use as "evidence" if finger-pointing does occur, and
  - Participate in planning and design of system tests to ensure that software is adequately tested.
- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.
- Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

System Testing

- Recovery Testing
  - Many computer based systems must recover from faults and resume processing within a pre-specified time.
  - In some cases, a system must be fault tolerant, i.e., processing faults must not cause overall system function to cease.
  - In other cases, a system failure must be corrected within a specified period of time or severe economic damage will occur.
  - Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.
  - If recovery is automatic (performed by the system itself), re-initialization, check-pointing mechanisms, data recovery, and restart are evaluated for correctness.
  - If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

System Testing

- Security Testing
  - Any computer-based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration.
  - Penetration spans a broad range of activities: hackers who attempt to penetrate systems for sport; disgruntled employees who attempt to penetrate for revenge; dishonest individuals who attempt to penetrate for illicit personal gain.
  - Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.
  - The system's security must, of course, be tested for invulnerability from frontal attack—but must also be tested for invulnerability from flank or rear attack.
  - During security testing, the tester plays the role(s) of the individual who desires to penetrate the system.
  - The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to breakdown any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry.

System Testing

- Stress Testing
  - Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example,
    - Special tests may be designed that generate ten interrupts per second, when one or two is the average rate.
    - Input data rates may be increased by an order of magnitude to determine how input functions will respond.
    - Test cases that require maximum memory or other resources are executed.
    - Test cases that may cause thrashing in a virtual operating system are designed.
    - Test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.
  - A variation of stress testing is a technique called **sensitivity testing**.
  - In some situations (the most common occur in mathematical algorithms), a very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation.
  - Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

System Testing

- Performance Testing
  - For real-time and embedded systems, software that provides required function but does not conform to performance requirements is unacceptable.
  - Performance testing is designed to test the run-time performance of software within the context of an integrated system.
  - Performance testing occurs throughout all steps in the testing process.
  - Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted.
  - However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.
  - Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation.
  - It is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion.
  - External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and sample machine states on a regular basis.
  - By instrumenting a system, the tester can uncover situations that lead to degradation and possible system failure.

Debugging

- Software testing is a process that can be systematically planned and specified.
- Test case design can be conducted, a strategy can be defined, and results can be evaluated against prescribed expectations.
- Debugging occurs as a consequence of successful testing.
- When a test case uncovers an error, debugging is the process that results in the removal of the error.
- Although debugging can and should be an orderly process, it is still very much an art.
- A software engineer, evaluating the results of a test, is often confronted with a "symptomatic" indication of a software problem.
- The external manifestation of the error and the internal cause of the error may have no obvious relationship to one another.
- The poorly understood mental process that connects a symptom to a cause is debugging.

Debugging

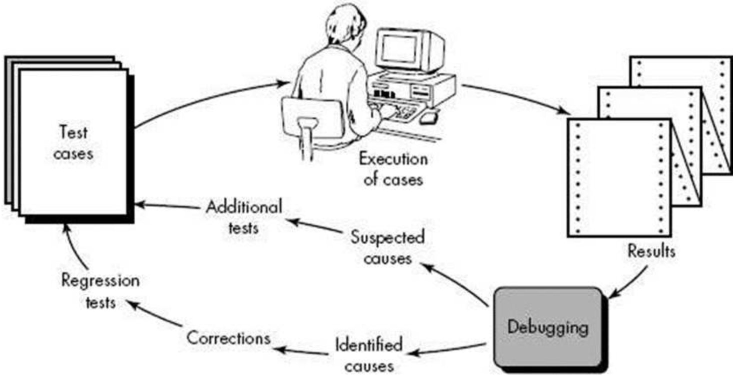

Figure 6: Debugging Process

Debugging

- The Debugging Process
  - Debugging is not testing but always occurs as a consequence of testing.
  - The debugging process begins with the execution of a test case.
  - Results are assessed and a lack of correspondence between expected and actual performance is encountered.
  - In many cases, the non corresponding data are a symptom of an underlying cause as yet hidden.
  - The debugging process attempts to match symptom with cause, thereby leading to error correction.
  - The debugging process will always have one of two outcomes:
    - The cause will be found and corrected, or
    - The cause will not be found.
  - In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.
  - During debugging, we encounter errors that range from mildly annoying (e.g., an incorrect output format) to catastrophic (e.g. the system fails, causing serious economic or physical damage).
  - As the consequences of an error increase, the amount of pressure to find the cause also increases.

Debugging Approaches

- Brute Force
  - Most common and least efficient method for isolating the cause of a software error.
  - Apply brute force debugging methods when all else fails.
  - Using a “let the computer find the error” philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements.
  - We hope that somewhere in the morass of information that is produced we will find a clue that can lead us to the cause of an error.
  - Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time.
- Backtracking
  - 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 backward (manually) until the site of the cause is found.
  - Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.

Debugging Approaches

- Cause Elimination
  - Manifested by induction or deduction and introduces the concept of binary partitioning.
  - Data related to the error occurrence are organized to isolate potential causes.
  - A "cause hypothesis" is devised and the aforementioned data are used to prove or disprove the hypothesis.
  - Alternatively, a list of all possible causes is developed and tests are conducted to eliminate each.
  - If initial tests indicate that a particular cause hypothesis shows promise, data are refined in an attempt to isolate the bug.

Software Quality Assurance

- Quality
  - It is a measurable quantity which can be compared with known standards for any product or process.
  - It specifies the degree to which a product or process maintains the known standards.
  - Quality can be classified into two groups:
    - **Quality of design:** It refers to the standards maintained during design processes which include the grade of i) Materials, ii) Tolerances and iii) Performance specifications.
    - **Quality of conformance:** It refers to the degree to which the design specifications are followed during manufacturing.
- Quality Control
  - It is a series of inspections, reviews and tests used throughout the development cycle to ensure that each work product meets the requirements placed up on it.
  - It includes a feedback loop to the process that created the work product. The feedback loop helps tune up the process during the creation of the product.
  - It should be a part of the manufacturing process.

Software Quality Assurance

- Quality Assurance
  - It is the auditing and reporting functions of the management.
  - If the data provided through quality assurance identify problems then it is the management's responsibility to solve the problems and apply the necessary resources to resolve the quality issues.
  - The goal of quality assurance is to provide management to provide information and data related to product quality so that they can evaluate whether or not the manufacturing process is maintaining the product quality.

Software Quality Assurance

- Costs related to maintenance of Quality
  - **Prevention Cost:** Cost required to implement Quality planning, Formal technical reviews, Test equipment, and Training.
  - **Appraisal Cost:** It includes activities to gain the quality standard of a product when going through the process for the first time.
    - In-process and inter process inspections;
    - Testing; and
    - Equipment calibration and maintenance.
  - **Failure Cost:** These costs are raised when a failure occurs otherwise these costs get avoided.
    - *Internal failure costs:* These are incurred when the defects are detected before the product gets delivered to the customer. (Rework, Repairs, and Failure mode analysis)
    - *External failures:* These are the costs incurred when the defects are detected after the product has reached the user. (Complaint resolution, Product return and replacement, Warranty support, etc)

Software Quality Assurance Activities

- The activities performed in software quality assurance involves two groups:
  - The software engineers doing the technical work. Their work includes:
    - Applying technical methods and measures for ensuring high quality;
    - Conducting formal reviews; and
    - Performing well planned software testing.
  - The *SQA group* that has the responsibility for
    - Quality assurance planning,
    - Oversight,
    - To assist the software engineering team in achieving high standards of production,
    - Record keeping,
    - Analysis, and
    - Reporting.
- The SQA activities performed by both SQA groups and Software engineering group are governed by an SQA plan for the project.

Software Quality Assurance Activities

- An SQA plan consists of the following activities:
  - **Management section:** It consists of the following standards:
    - The organizational structure for the project,
    - Tasks and activities to be maintained in the organizational structure,
    - The organizational roles and responsibilities for maintaining good product quality.
  - **Documentation section:** It describes the standards to be maintained in the following elements:
    - Project documents (e.g. project plan),
    - Models (e.g. class hierarchies),
    - Technical documents (e.g. test plans, specifications),
    - User documents (e.g. help files).
  - **Standards, Practices and Conventions section:** It contains the list of applicable standards and practices to be followed during the software process. It requires that all project, process and product matrices that are used in the project should be listed out.
  - **Reviews and Audits section:** It describes the quality reviews and quality audits that are to be conducted by the software engineering group, the SQA group and the customer.

Software Quality Assurance Activities

- An SQA plan consists of the following activities:
  - **Test section:** It refers to the software test plan and testing procedures. It also refers to the standards to be maintained for keeping the records of the tests performed.
  - **Problem reporting and corrective action section:** It defines the procedures for reporting, tracking and resolving errors & defects. It also specifies the organizational responsibilities for these activities.

Statistical Quality Assurance

- Information about software defects is collected and categorized.
- An attempt is made to trace the cause of each defect to its basis.
- Using the Pareto principle to find out the vital view which states that 80 percent of the defects can be traced to 20 percent of all possible causes.
- It means that if all the errors are found and all the possible causes for those errors are also found, then its observed that about 20 percent of the possible causes would justify about 80 percent of the total errors. This 20 percent is collectively called the "vital few".
- Once the vital view causes are identified then the procedure of error correction should be carried out.

Software Maintenance

- Software Maintenance is a very broad activity that includes error, corrections, enhancement of capabilities, deletion of obsolete capabilities and optimization.
- Because change is inevitable, mechanisms must be developed for evaluating, controlling and making modifications.
- Thus, any work done to change the software after it is in operation is considered to be maintenance.
- The purpose is to preserve the value of software overtime.
- The value can be enhanced by expanding the customer base, meeting additional requirements, becoming easier to use, more efficient and employing newer technology.
- Maintenance may span for 500 years, whereas development may be 1–2 years.

Types of Maintenance

- Corrective Maintenance
  - This refers to modifications initiated by defects in the software.
  - A defect can result from, design errors, logic errors and coding errors.
  - **Design errors** occur when, changes made to the software are incorrect, incomplete, wrongly communicated or the change request is misunderstood.
  - **Logic errors** result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow or incomplete test data.
  - **Coding errors** are caused by incorrect implementations of detailed logic design and incorrect use of the source code logic.
  - Defects are also caused by data processing errors and system performance errors.
  - In the event of system failure due to an error, actions are taken to restore operation of the software system.
  - Due to pressure from management, maintenance personnel sometimes resort to emergency fixes known as patching.
  - The adhoc nature of this approach often gives rise to a range of problems that include increased program complexity and unforeseen ripple effects.
  - Unforseen ripple effects imply that a change to one part of a program may affect other sections in an unpredictable manner.
  - This is often due to lack of time to carry out a through "impact analysis" before effecting the change.

Types of Maintenance

- Adaptive Maintenance
  - It includes modifying the software to match changes in the over changing environment.
  - The term *environment* refers to the totality of all conditions and influences which act from outside upon the software.
  - For example, business rules, government policies, work patterns, software and hardware operating platforms.
  - A change to the whole or part of this environment will require corresponding modifications of the software.
  - Thus, this type of maintenance includes any work initiated as a consequence of moving the software to a different hardware or software platform – compiler, operating system or new processor.
  - Any change in the government policy can have far reaching ramification on the software.
  - When European countries had decided for "single European currency", this change affected all banking system software and was modified accordingly.

Types of Maintenance

- Perfective Maintenance
  - It means improving processing efficiency or performance, or restructuring the software to improve changeability.
  - When the software becomes useful, the user tends to experiment with new cases beyond the scope for which it was initially developed.
  - Explosion in requirements can take form of enhancement of existing system functionality or improvement in computational efficiency.
  - **Example:** providing a Management Information System with a data entry module or a new message handling facility.

Types of Maintenance

- Preventive Maintenance
  - There are long term effects of corrective, adaptive and perfective changes.
  - This leads to increase in the complexity of the software, which reflects deteriorating structure.
  - The work is required to be done to maintain it or to reduce it, if possible.
  - This work may be named as preventive maintenance.
  - This term is often used with hardware systems and implies such things as lubrication of parts before need occurs, or automatic replacement of banks of light bulbs before they start to individually burn out.