SOFTWARE ENGINEERING

(CSE3005)

9/21/2019

Prof. Anand Motwani

Faculty, SCSE

VIT Bhopal University

Unit IV

IMPLEMENTATION AND TESTING

Software Implementation Techniques: Coding Practices – Refactoring - Software Testing Fundamentals - Types of Testing: Unit, Integration and System testing - Testing Strategies: Black box and White box testing - System testing and debugging.

1. Software Implementation Techniques (Introduction):

- **Best coding practices** are a set of informal rules that the software development community has learned over time which can help improve the quality of software.
- Many computer programs remain in use for far longer than the original authors ever envisaged (sometimes 40 years or more) so any rules need to facilitate both initial development and subsequent maintenance and enhancement by people other than the original authors.
- Ninety-ninety rule, Tim Cargill is credited with this explanation as to why programming projects often run late.
- The size of a project or program has a significant effect on error rates, programmer productivity, and the amount of management needed
    - Maintainability.
    - Dependability.
    - Efficiency.
    - Usability.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

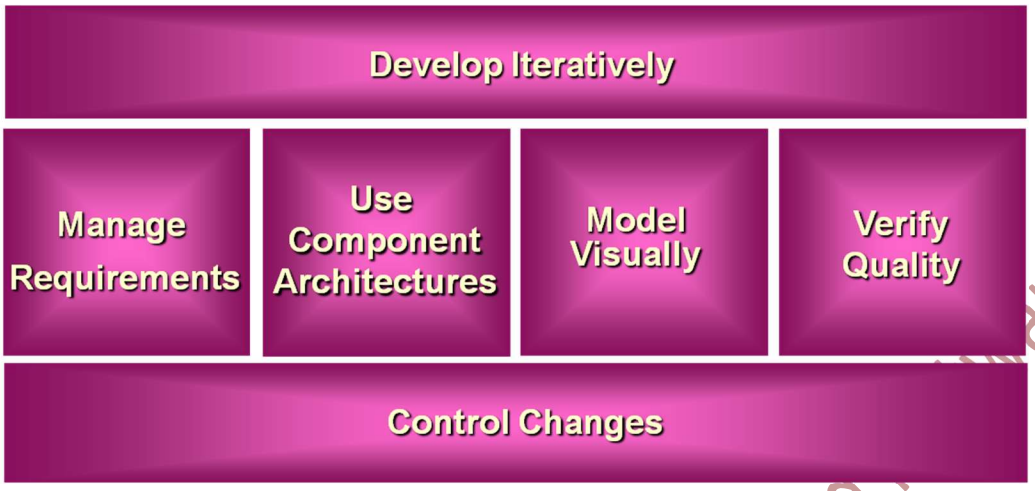


Fig. 1. Best Practices of Software Engineering

1.1 Coding Practices

Refactoring

- Refactoring is usually motivated by noticing a code smell. For example the method at hand may be very long, or it may be a near duplicate of another nearby method. Once recognized, such problems can be addressed by *refactoring* the source code, or transforming it into a new form that behaves the same as before but that no longer "smells".
- For a long routine, one or more smaller subroutines can be extracted; or for duplicate routines, the duplication can be removed and replaced with one shared function. Failure to perform refactoring can result in accumulating technical debt.
- There are two general categories of benefits to the activity of refactoring.
    - Maintainability
    - Extensibility
- Before applying a refactoring to a section of code, a solid set of automatic unit tests is needed. The tests are used to demonstrate that the behavior of the module is correct before the refactoring.
- If it inadvertently turns out that a test fails, then it's generally best to fix the test first, because otherwise it is hard to distinguish between failures introduced by refactoring and failures that were already there. After the refactoring, the tests are run again to verify the refactoring didn't break the tests.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

- Of course, the tests can never prove that there are no bugs, but the important point is that this process can be cost-effective: good unit tests can catch enough errors to make them worthwhile and to make refactoring safe enough.

1.2 Software Testing Fundamentals

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data. You check the results of the test run for errors, anomalies, or information about the program's non-functional attributes.

The testing process has two distinct goals:

1. To demonstrate to the developer and the customer that the software meets its requirements. For custom software, this means that there should be at least one test for every requirement in the requirements document. For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.

- This first goal leads to validation testing, where you expect the system to perform correctly using a given set of test cases that reflect the system's expected use.

2. To discover situations in which the behavior of the software is incorrect, undesirable, or does not conform to its specification. These are a consequence of software defects. Defect testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations, and data corruption.

- The second goal leads to defect testing, where the test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

Of course, there is no definite boundary between these two approaches to testing. During validation testing, you will find defects in the system; during defect testing, some of the tests will show that the program meets its requirements.

Edsger Dijkstra, an early contributor to the development of software engineering, eloquently stated (Dijkstra et al., 1972):

"Testing can only show the presence of errors, not their absence."

- Testing is part of a broader process of software verification and validation (V & V).
- Verification and validation are not the same thing, although they are often confused.
- Barry Boehm, a pioneer of software engineering, succinctly expressed the difference between them (Boehm, 1979):
- ■ 'Validation: Are we building the right product?'
- ■ 'Verification: Are we building the product right?'

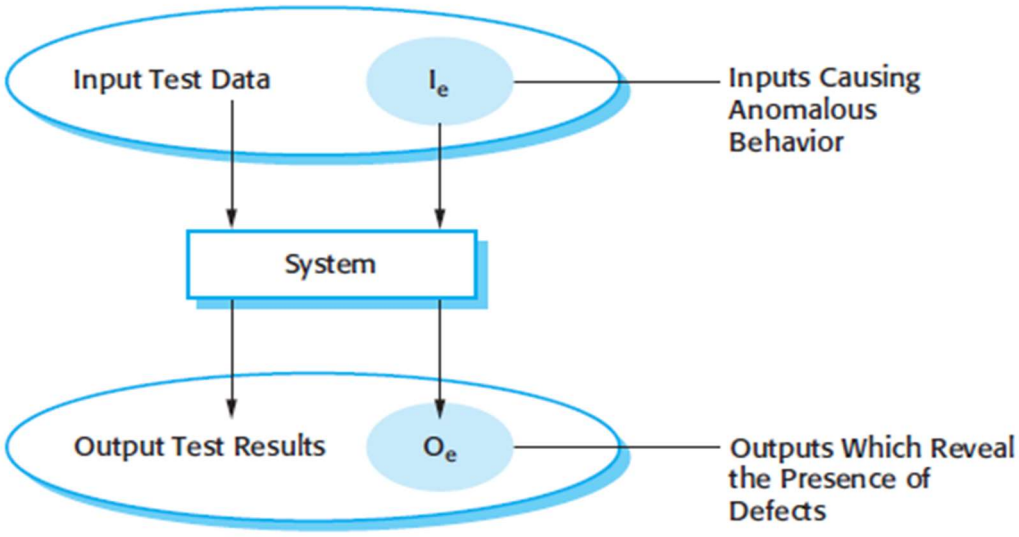


Fig. 2.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

Test planning is concerned with scheduling and resourcing all of the activities in the testing process. It involves:

- Defining the testing process, taking into account the people and the time available. Usually, a test plan will be created, which defines what is to be tested, the predicted testing schedule, and how tests will be recorded.
- For critical systems, the test plan may also include details of the tests to be run on the software.

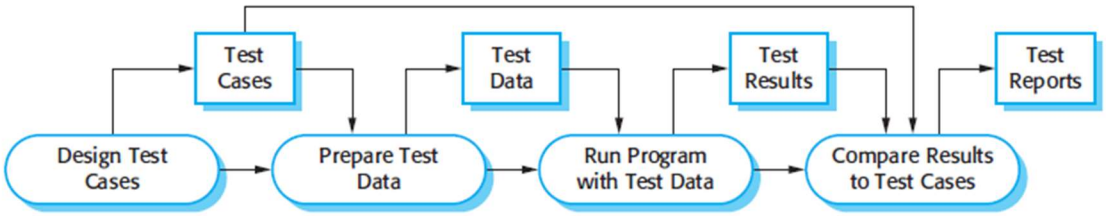


Fig. 3. Test plan

1.3 More testing fundamentals (*System testing and debugging)*

Normally, programmers carry out some testing of the code they have developed. This often reveals program defects that must be removed from the program. This is called **debugging**.
Defect testing and debugging are different processes. *Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.*

Debugging process: When you are debugging, you have to generate hypotheses about the observable behavior of the program then test these hypotheses in the hope of finding the fault that caused the output anomaly. Testing the hypotheses may involve tracing the program code manually. It may require new test cases to localize the problem. Interactive debugging tools, which show the intermediate values of program variables and a trace of the statements executed, may be used to support the debugging process.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

1.4 Stages and Types of Testing:

Except for small programs, systems should not be tested as a single, monolithic unit. Figure below shows a three-stage testing process in which system components are tested then the integrated system is tested and, finally, the system is tested with the customer's data. Ideally, component defects are discovered early in the process, and interface problems are found when the system is integrated. However, as defects are discovered, the program must be debugged and this may require other stages in the testing process to be repeated. Errors in program components, say, may come to light during system testing. The process is therefore an iterative one with information being fed back from later stages to earlier parts of the process.

The stages in the testing process are:

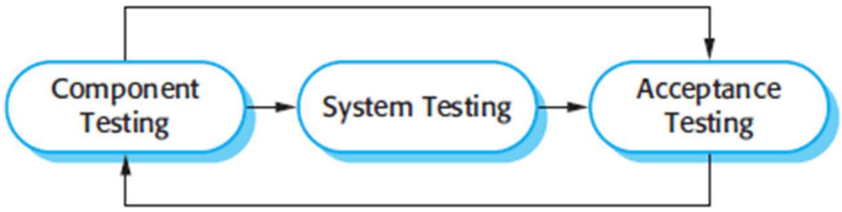


Fig. 4. Stages of Testing

1. Development (Unit & Component) testing: The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities. Test automation tools, such as JUnit (Massol and Husted, 2003), that can re-run component tests when new versions of the component are created, are commonly used.

During development, testing may be carried out at three levels of granularity:

1. Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods. Unit testing is the

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

process of testing program components, such as methods or object classes. Individual functions or methods are the simplest type of component. Your tests should be calls to these routines with different input parameters. Example of Unit testing is given at the end of the document.

2. Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.

3. System Testing: Refer below

2. System testing: System components are integrated to create a complete system. The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems. It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties. For large systems, this may be a multi-stage process where components are integrated to form subsystems that are individually tested before these sub-systems are themselves integrated to form the final system.

- *This testing is conducted on a complete, integrated system to evaluate the system's compliance or reliability with its specified requirements.*

3. Acceptance testing: This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition, because the real data exercise the system in different ways from the test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.
Another definition:

- *It is a Formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria.*

Acceptance testing is sometimes called 'alpha testing'. Custom systems are developed for a single client. The alpha testing process continues until the system developer and the client

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

agree that the delivered system is an acceptable implementation of the requirements. ***Alpha*** *Testing is conducted by a team of highly skilled testers at development site*

When a system is to be marketed as a software product, a testing process called 'beta testing' is often used.

Beta testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors that may not have been anticipated by the system builders. After this feedback, the system is modified and released either for further beta testing or for general sale.

- *Beta Testing is always conducted in Real Time environment by customers or end users at their own site.*

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

1.5 SDLC – V model

- The V-model is a type of SDLC model where process executes in a sequential manner in V-shape.
- It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage.
- Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it.

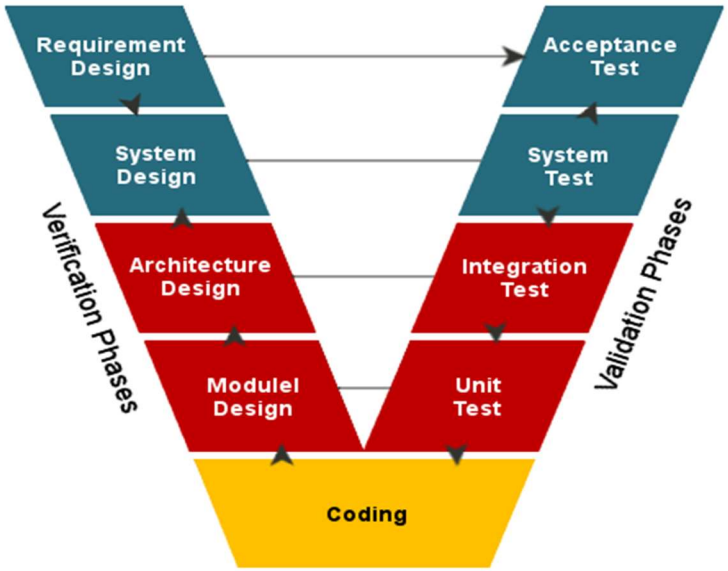


Fig. 5. SDLC-V model

- *In this execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation Model. V-Model is an extension of the Waterfall Model and is based on association of a testing phase for each corresponding development stage.*

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

Figure below is another illustration of how test plans are the link between testing and development activities. This is sometimes also called the V-model of development (turn it on its side to see the V).

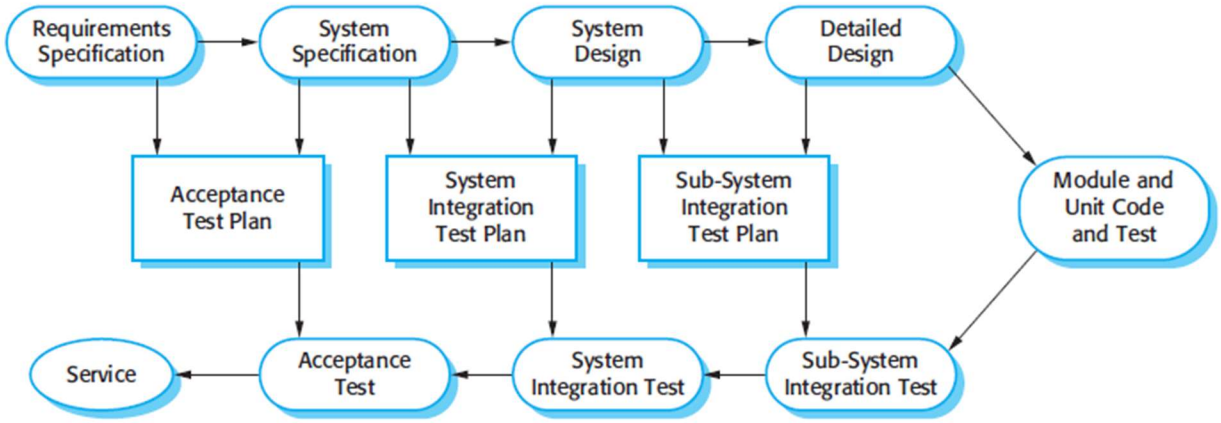


Fig. 6. Testing phases in a plan-driven software process

Other Important Testing types and strategies:

- **Regression Testing:** Retesting of a software system to confirm that changes made to few parts of the codes has not any side effects on existing system functionalities. The emphasis on regression testing in agile methods lowers the risk of introducing new errors through refactoring. Regression testing involves running test sets that have successfully executed after changes have been made to a system. The regression test checks that these changes have not introduced new bugs into the system and that the new code interacts as expected with the existing code. Regression testing is very expensive and often impractical when a system is manually tested, as the costs in time and effort are very high. In such situations, you have to try and choose the most relevant tests to re-run and it is easy to miss important tests. Automated tests and a testing framework, such as JUnit, radically simplify regression testing as the entire test set can be run automatically each time a change is made.

Note: All the above testing types come under the level of Functional Testing. Another level of testing is given below:

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

- **Non-Functional Testing:** *Non-Functional testing is designed to figure out if your product will provide a good user experience.*
    - ***Performance Testing:*** *To evaluate the performance of components of a particular system under a particular workload.*
    - ***Load Testing:*** *Testing the behaviour of the system under a specific load or to get the breakeven point where system starts downgrading it's performance.*
    - ***Stress Testing:*** *It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.*
    - ***Usability Testing:*** *Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.*
    - ***Security Testing:*** *This intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.*
    - ***Portability Testing:*** *Software reliability is the probability that software will work properly in a specified environment and for a given amount of time.*

1.6 Testing Strategies

- **Black-box testing:** An approach to testing where the testers have no access to the source code of a system or its components. The tests are derived from the system specification. Release testing is usually a black-box testing process where tests are derived from the system specification.

- **White-box or structural testing**:

  It is an approach to program testing where the tests are based on knowledge of the structure of the program and its components. Access to source code is essential for white-box testing. It is a systematic approach to testing where knowledge of the program source code is used to design defect tests. The aim is to design tests that provide some

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

level of program coverage. That is, the set of tests should ensure that every logical path through the program is executed, with the consequence that each program statement is executed at least once.

1.7 Ways to perform Testing

1. Manual Testing: Test Cases executed manually. In this method the tester plays an important role of end user and verifies that all the features of the application are working correctly. The tester manually executes test cases without using any automation tools. The tester prepares a test plan document which describes the detailed and systematic approach to testing of software applications. Test cases are planned to cover almost 100% of the software application. As manual testing involves complete test cases it is a time consuming test.

2. Automation (Automated) Testing: Testing performed with the help of automation tools. (Most popular testing automation tool is Selenium). Agile methods recommend that very frequent system builds should be carried out with automated testing to discover software problems.

1.8 Test Artifacts

- Test Basis: It is the information needed in order to start the test analysis and create our Test Cases.
- Test Case Specification: A document described detailed summary of what scenarios will be tested, how they will be tested, how often they will be tested.
- Test Scenario: It is also called Test Condition or Test Possibility means any functionality that can be tested.
- Test Case: A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- Test Script - Is a collection of test cases for the software under test (manual or automated)

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

Test case Example:

- To test a method called 'catWhiteSpace' in a 'Paragraph' object that, within the paragraph, replaces sequences of blank characters with a single blank character. Identify testing partitions for this example and derive a set of tests for the 'catWhiteSpace' method.

Solution:

Testing partitions / scenarios are:

- Strings with only single blank characters
- Strings with sequences of blank characters in the middle of the string
- Strings with sequences of blank characters at the beginning/end of string

Examples of tests:

- The quick brown fox jumped over the lazy dog (only single blanks)
- The quick brown fox jumped over the lazy dog (different numbers of blanks in the sequence)
- The quick brown fox jumped over the lazy dog (1st blank is a sequence)
- The quick brown fox jumped over the lazy dog (Last blank is a sequence)
- The quick brown fox jumped over the lazy dog (2 blanks at beginning)
- The quick brown fox jumped over the lazy dog (several blanks at beginning)
- The quick brown fox jumped over the lazy dog (2 blanks at end)
- The quick brown fox jumped over the lazy dog (several blanks at end)
- Etc.

Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University

Example Test Script – 1:

- System Test of input of numeric month into data field

Ref.	Field/Button	Action	Input	Expected Result	Pass/Fail
001	Month	Enter Data	0	Data rejected. Error Message 'Invalid Month'	Fail
002	Month	Enter Data	1	Data Accepted, January Displayed	Pass
003	Month	Enter Data	06	Data Accepted, June Displayed	Pass
004	Month	Enter Data	12	Data Accepted, December Displayed	Pass
005	Month	Enter Data	13	Data rejected. Error Message 'Invalid Month'	Fail

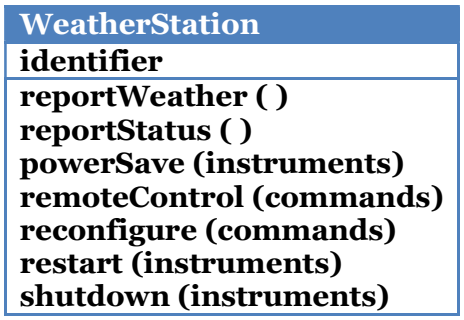
Unit testing Example:

When you are testing object classes, you should design your tests to provide coverage of all of the features of the object. This means that you should:

• test all operations associated with the object;

• set and check the value of all attributes associated with the object;

• put the object into all possible states. This means that you should simulate all events that cause a state change.



Prepared by:
Prof. Anand Motwani
Faculty SCSE,
VIT Bhopal University