CHAPTER-8

SOFTWARE TESTING AND COST ESTIMATION

-Ela Thakur

Software testing is the process of exercising with the specific intent of finding errors prior to delivery to the end user. Testing is part of a broader process of software verification and validation. Testing results in higher quality software, more satisfied user and lower maintenance cost, more accurate and reliable results. Testing costs 1/3 to ½ of the total cost of software development process.

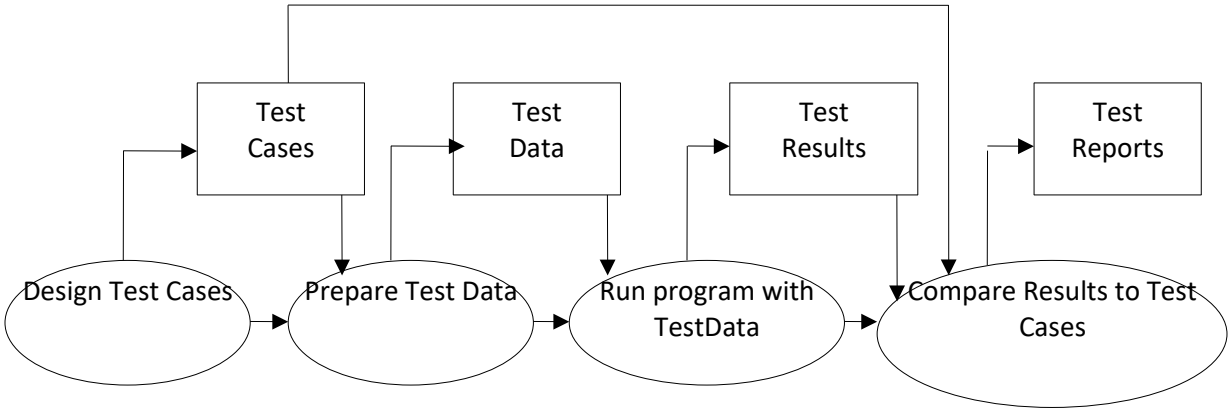


Fig: A model of the software testing process

8.1. SYSTEM TESTING:

System testing fully exercise the computer based system to verify the system elements have been properly integrated and perform allocated functions. An independent testing team is responsible for system testing. The tests are based on system specification.

There are two phases in system testing:

a) Integration testing: (IOE Q:Explain Integration testing 067 asadh)
   In this type of testing the test team has access to the system code. The system is tested as components are integrated.
   The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.
   Some different types of integration testing are big bang, top-down, and bottom-up.
   Big Bang
   In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.
   Top-down and Bottom-up
   Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.
   All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for

integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

b) Release testing:

In this type of testing a separate testing team tests a complete version of the system before it is released to users. System testing by the development team should focus on discovering bugs in the system. The aim of release testing is to check that the system meets the requirements of system stakeholders. System testing by the development team should focus on discovering bugs in the system. Release testing is usually a **black-box testing** process where tests are derived from the system specification. The system is treated as black-box whose behavior. Another name for this is 'functional testing', so called because the tester is only concerned with functionality and not the implementation of the software.

8.2. COMPONENT TESTING:

Several individual units are integrated to create composite components. Component testing should focus on testing component interfaces. It is a defect testing process. The components may be:

- Individual function or methods within an object
- Object classes with several attributes and methods
- Composite components with defined interfaces used to access their functionality

Fig: Testing Phases

8.3. TEST CASE DESIGN:

Test case design involves designing the text cases (inputs and outputs) used to test the system. The goal of test case design is to create a set of tests that are effective in validation and defect testing.

❖ Design approaches:

- Requirement-based testing:
  Used in validation testing technique where we consider each requirement and tests for that requirement.
- Partition Testing:
  It is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.
- Structural Testing(White-Box Testing):
  It is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

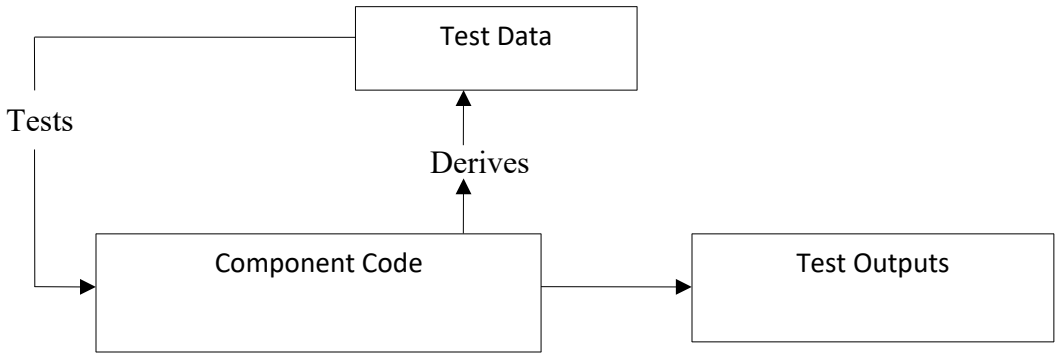


Fig: Structural Testing

8.4. TEST AUTOMATION:

In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes to predicted outcomes. Test automation can automate previous repetitive but necessary testing in a formalized testing process already in place, or add additional testing that would be difficult to perform manually. It reduces testing costs by supporting the test process with range of software tools. System such as 'Junit' supports the automatic execution of tests. There are two general approaches to test automation:

- **Code-driven testing**. The public (usually) interfaces to classes, modules or libraries are tested with a variety of input arguments to validate that the results that are returned are correct.
- **Graphical user interface testing**. A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.

8.5. METRICS FOR TESTING:

Majority of metric for testing proposed focus on the process of testing ,not the technical characteristics of the test themselves.

- Halstead metrics applied to testing:
  Testing effort can be estimated using metrics derived from Halstead measures.
  PL=1/ [(n1/2)*(N2/n2)]
  e=V/ PL
  Where; PL is program level
  e is Halstead effort
  V is program volume
  n1 is no. of distinct operations that appears in program
  n2 is the no. of distinct operands that appears in a program
  N2 is the total no. of operand occurrence
  The percentage of overall testing effort to be allocated to a module 'K' can be estimated as:
  $K=e(K)/\sum e(i)$ where; e(K) is computed for module K
  And; $\sum e(i)$ is the sum of effort across all modules of the system

8.6. SOFTWARE PRODUCTIVITY:

Software productivity is the ratio between the amount of software produced to the labor and expense of producing it. There are two measures of software productivity:

I. Function-related measures:
   Productivity is expressed in terms of the amount of useful functionality produced in some given time. Function point in a program is computed by measuring program features:-
   A. External inputs and outputs
   B. User interactions
   C. External interfaces
   D. Files used by the system

II. Size:
   - Line of code delivered

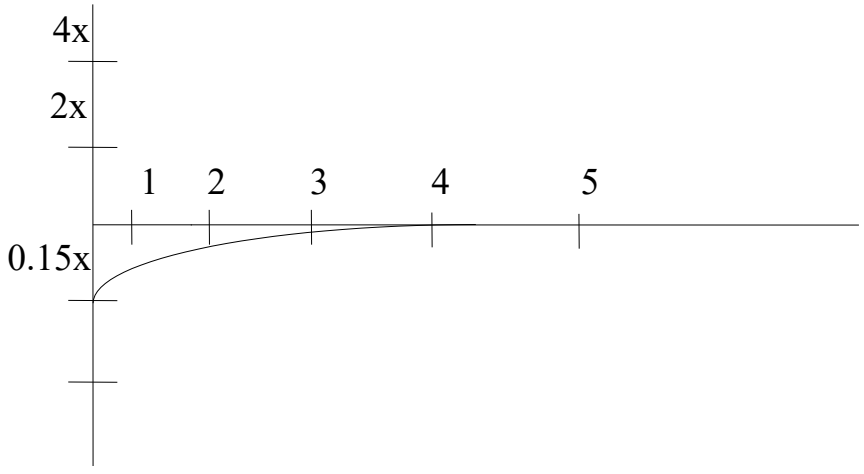
- Also measure no. of delivered object code instruction or no. of pages of system documentation
- Useful for programming in FORTRAN, Assembly or COBOL
- More expressive the programming language, the lower apparent productivity
  Example: A system which might be coded in 5000 lines of assembly code. The development time for the various phases in 28 weeks,
  Then; productivity= (5000/28) *4
  =714 lines/ month

8.7. ESTIMATION TECHNIQUES

There is no simple way to make an accurate estimate of the effort required to develop a software system. Initial estimates are based on inadequate information in a user requirement definition. People in the project may be unknown. Project cost estimates may be self-fulfilling. The estimate defines the budget and the product is adjusted to meet the budget.

Some estimation techniques are:

- Algorithm cost modeling
- Expert judgment
- Estimation by analogy
- Parkinson's law
- Pricing to win



0.25x

1: Feasibility
2: Requirement Design
3: Code
4: Delivery

Fig: Estimation Uncertainty

8.8. ALGORITHM COST MODELLING

Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project manager.
Effort = A* size^B*M
Where; A is an organization-dependent constant
B reflects the disproportionate effort for large project
M is a multiplier reflecting product, process and people attributes

❖ COCOMO MODEL(CONSTRUCTIVE COST MODEDL):

This is an empirical model that was derived by collecting data from a large number of software projects. This is a well-documented and non-proprietary estimation model.

Formula for effort computation for system prototype:
PM= [NAP*(1 - %reuse/100)] / [PROD]
Where; PM is effort estimation in person-motion
NAP is the total no. of application points in the delivered system
%reuse is an estimation of the amount of reused code in the development
PROD is the application-point productivity

8.9. PROJECT DURATION AND STAFFING:

As well as effort estimation, managers must estimate the calendar time required in completing a project and the staff required.

Calendar time estimation (COCOMO)
TDEV = 3*(PM) ^ [(0.33+0.2)*(B – 1.01)]
Where; PM is the effort computation
B is the exponent computed (B is 1 for the early prototyping mode)

Time required is independent of the number of people working in the project. Staff required cannot be computed by dividing the development time by the required schedule. The number of people working in a project varies depending on the phase of the project.

✓ IOE QUESTIONS:

- What is regression testing?

➢ A test suite is developed incrementally as a program is developed. We can always run regression tests to check the changes to the program have not introduced new bugs.
The intent of regression testing is to ensure that a change has not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.

Common methods of regression testing include rerunning previously-completed tests and checking whether program behavior has changed and whether previously-fixed faults have re-emerged. Regression testing can be used to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.

Regression testing can be used not only for testing the correctness of a program, but often also for tracking the quality of its output. For instance, in the design of a compiler, regression testing could track the code size, simulation time and compilation time of the test suite cases.

- What are the basic principles of software testing? List the characteristics of testability of software. List out possible errors of black-box testing.

➢ The basic principles of software testing are:

1) Testing an application exhaustively is impossible

   Example

   Assume that we have been given an application which produces bank statements that are sent to the customers. It is impossible for us to test each and every notice that is being generated to each customer. Only way to perform is to identify a suitable sample for it.

2) Testing is context based - Software testing is always based on the purpose to which the software built will be used.

   Example:

   An application built to be used inside an aircraft, requires rigorous testing and subject it to high quality standards. But an application built for storing the addresses in a Personal computer need not be tested rigorous similar to the previous application.

3) Testing a software is to find out the defects - not to prove that the software is error free.

   The main objective of testing a software is to find out as many defects as possible. Testing is not done to prove that the software is error free.

   Example

   A software may not have any reported defects (all defects identified are fixed) but still it may fail in the production environment.

4) Testing starts from requirements gathering, In other words early testing reduces the amount of money, rework involved etc.

   Testing must be started as early as the Testing life cycle begins. The earlier we identify and fix defects, the greater the money is saved.

5) The number of Defects in an application seems to come from one or few areas or modules of the application and not spread evenly.

   Example:
   The module was prepared by new programmer
   The complexity of that particular module is very high etc..

   6) Performing the similar kind of testing again and again does not identify the defects.

   Executing same set of test cases will not identify the defects present in the software.

   7) Absence of errors in an application does not mean that, the application is free from defects.

The characteristics of testability of software are:

- Operability
- Observability
- Controllability
- Decomposability
- Simplicity
- Stability
- Understandability

Possible errors of Black-box testing are:

- Only a small number of possible inputs can be tested and many program paths will be left untested
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design
- Tests can be redundant if the software designer/ developer has already run a test case

- Testing is one of the very important core parts of software development and implementation. Comment on this statement and explain various testing techniques.

➢ Software testing is the process of exercising with the specific intent of finding errors prior to delivery to the end user. Testing is part of a broader process of software verification and validation. Testing results in higher quality software, more satisfied user and lower maintenance cost, more accurate and reliable results. Testing costs 1/3 to ½ of the total cost of software development process. Hence testing is the very important core parts of software development and implementation.

Various testing techniques are:

A. SYSTEM TESTING:

System testing fully exercise the computer based system to verify the system elements have been properly integrated and perform allocated functions. An independent testing team is responsible for system testing. The tests are based on system specification.

There are two phases in system testing:

- Integration testing:
  In this type of testing the test team has access to the system code. The system is tested as components are integrated.
  The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block"

approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.
Some different types of integration testing are big bang, top-down, and bottom-up.

Big Bang

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

Top-down and Bottom-up

Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.
All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.
Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.
The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

- Release testing:
  In this type of testing a separate testing team tests a complete version of the system before it is released to users. System testing by the development team should focus on discovering bugs in the system. The aim of release testing is to check that the system meets the requirements of system

stakeholders. System testing by the development team should focus on discovering bugs in the system. Release testing is usually a **black-box testing** process where tests are derived from the system specification. The system is treated as black-box whose behavior. Another name for this is 'functional testing', so called because the tester is only concerned with functionality and not the implementation of the software.

B.COMPONENT TESTING:

Several individual units are integrated to create composite components. Component testing should focus on testing component interfaces. It is a defect testing process. The components may be:

- Individual function or methods within an object
- Object classes with several attributes and methods

- Composite components with defined interfaces used to access their functionality

Fig: Testing Phases

- What problems may be encountered when top down integration is chosen?

➢ The problems encountered are:

- The solution provides limited coverage in the first phases.
- A minimal percentage of user accounts are managed in the first phases.
- You might have to develop custom adapters at an early stage.
- The support and overall business will not realize the benefit of the solution as rapidly.
- The implementation cost is likely to be higher.

- Why does software project fail after it has passed through acceptance testing?

➢ The reasons are:

- Poor user input
- Stakeholder conflicts
- Vague requirements
- Late failure warning signals
- Communication breakdowns
- Hidden costs of going “Lean And Mean”

- Consider a program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and values may be from interval [0, 100]. The program output may have one of the following words. [Not a quadratic equation; Real root, Imaginary roots, Equal roots]. Design test cases to test this program.