Software Testing

A definition

Testing is a process of executing a program with the intent of finding errors

What world are we talking of?

Messaging

Multimedia

Embedded Systems

Software

Space

Hardware

Networking

You Name it

Security

Travel

Learning

Database

Telecommunications

Automobile

Analysis

Home

Medicine

Infotainment

What world are we talking of?

Messaging

Multimedia

Embedded Systems

Software is (nearly) everywhere. Thus we are talking about that "everything".

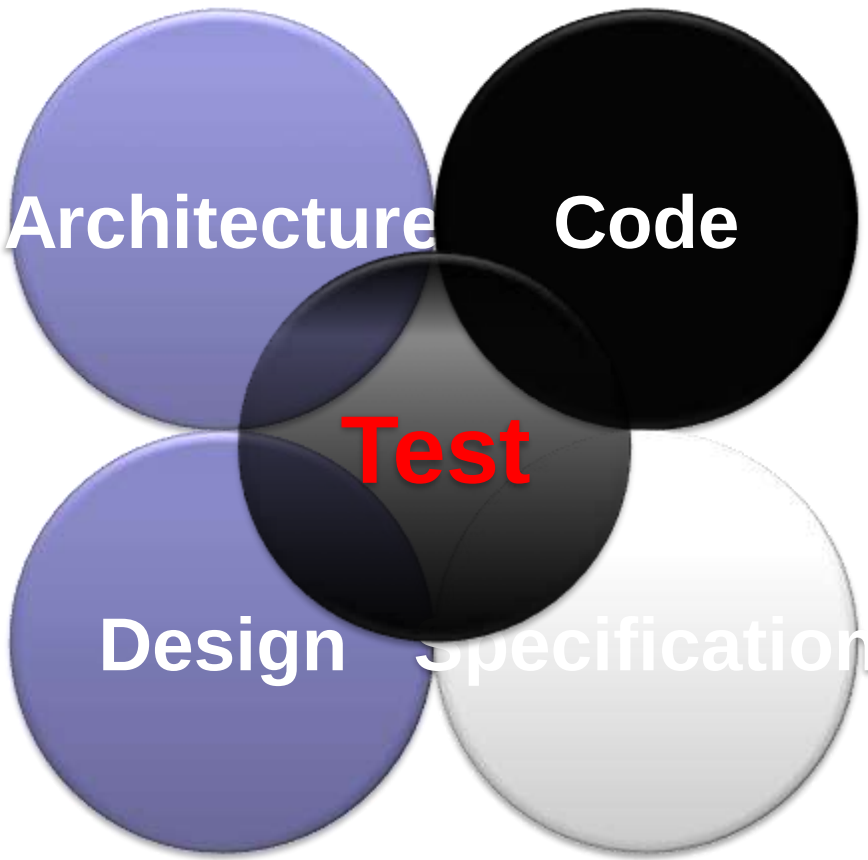
Home

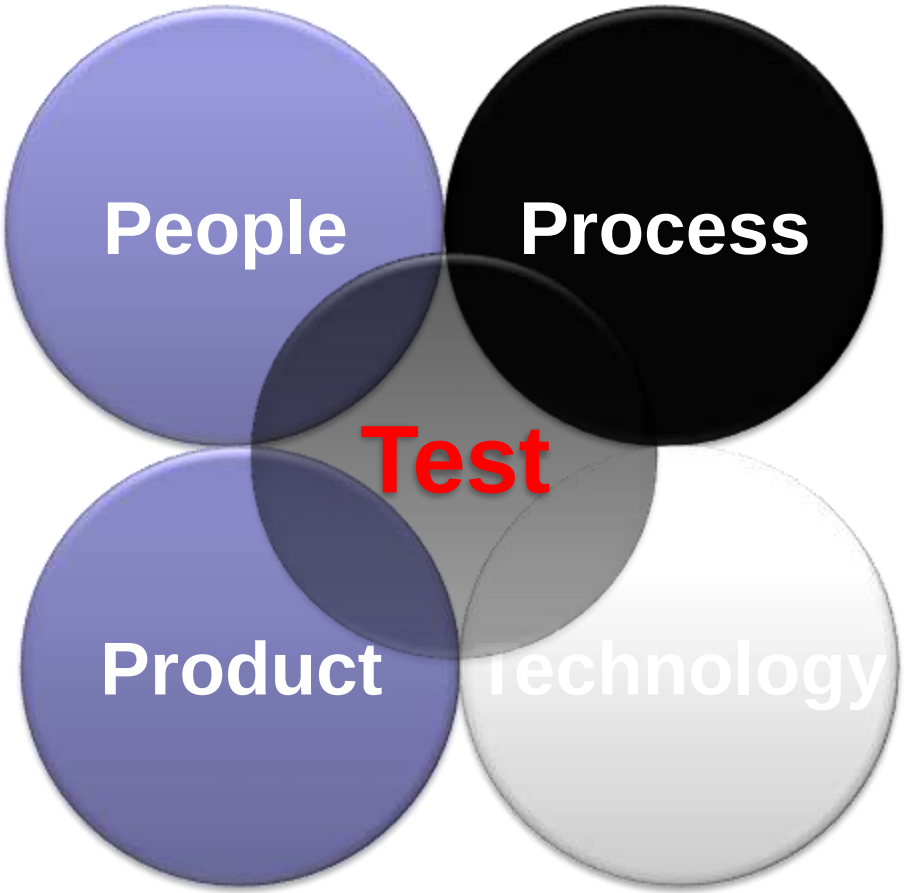
Infotainment

Medicine

Analysis

Building Blocks





Software Engineering

IEEE Definition

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of Software; that is the application of engineering to software (2) The study of approaches as in (1).

Software Engineering is the establishment and use of a sound engineering principle in order to obtain economically software that is reliable and works efficiently on real machine. **[Fritz Bauer]**

- **Where did Software Engineering come from?**
- **Rather why was there a thought of Software Engineering?**

Goals of Software Engineering

1. To improve quality of software
2. To improve the productivity of developers and software teams

And many more…

Software Quality Attributes

Types of attributes:

- **Static**: structured, maintainable, testable code as well as the availability of correct and complete documentation.
- **Dynamic**: software reliability, correctness, completeness, consistency, usability and performance

Attributes:

- Completeness
- Consistency
- Usability
- Performance

What if testing does not detect any errors

Either

- the software is high quality

Or

- the testing process is low quality

Metrics are required on our testing process if we are to tell which is the right answer.

Test Techniques

Based on Engineers experience and intuition

- Exploratory
- Ad-hoc

Specification Based Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Tables
- …

Code Based Techniques

- Control Flow
- Data Flow
- …

Test Techniques

Techniques based on nature of application

- Object Oriented
- Component-based
- Web-based
- GUI
- Protocol Conformance
- Real Time Systems

Usage based testing

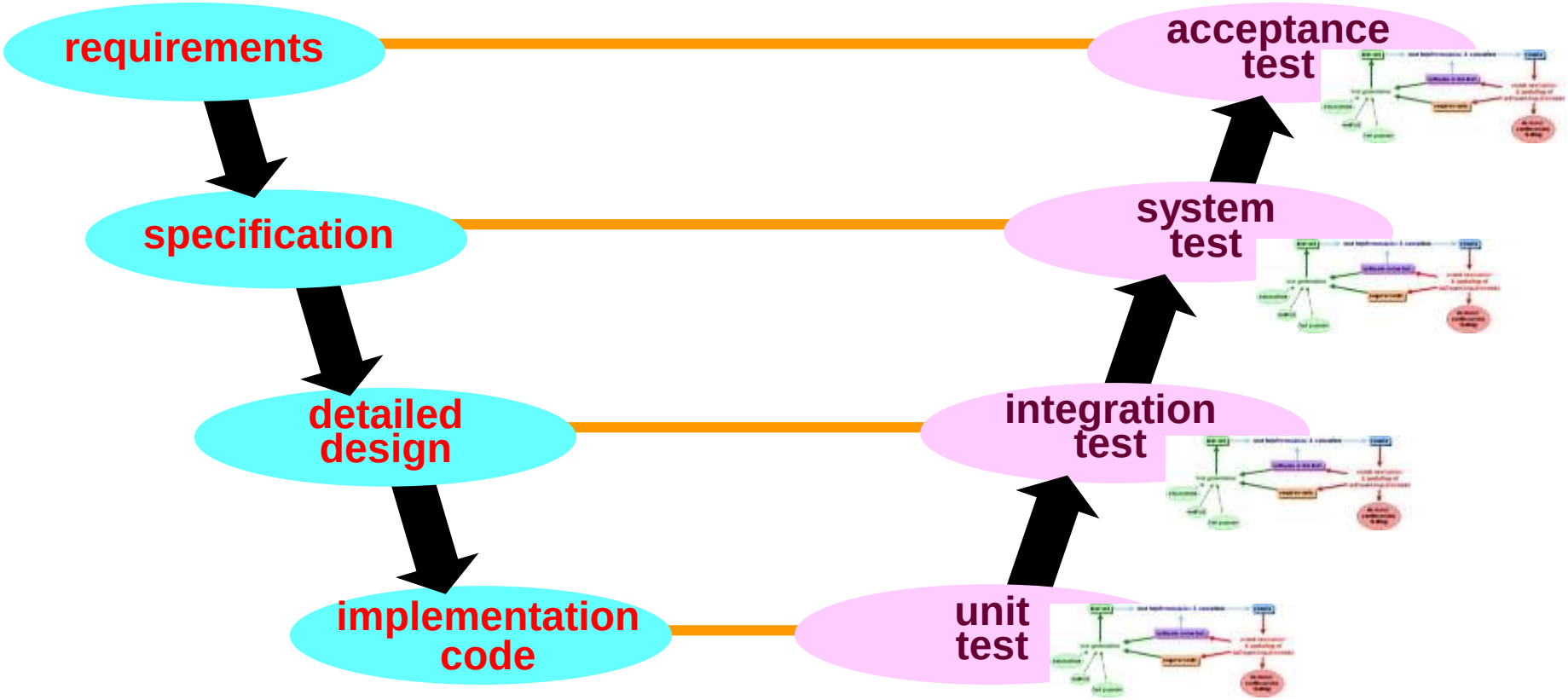
- Operational Profile
- Reliability Engineered Testing

Goals of SW Testing

- To show that the software system satisfies the requirements and performs as expected. The requirements may be explicit or implicit.
  - Explicit: User Interface, Specified Output
  - Implicit: Error handling, performance, reliability, security
- To have “confidence” in the software system. To assure and demonstrate that the Software works.
- To find defects
- To prevent defects
- Ensure software quality

Test process in software development

V-model:

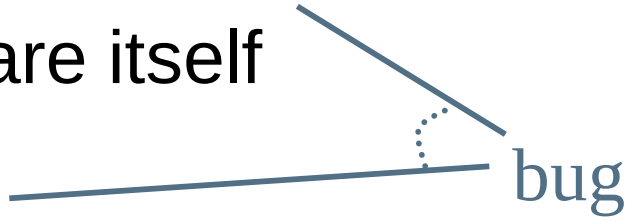


What is...

error: something wrong in software itself

fault: manifestation of an error

(observable in software behavior)



failure: something wrong in software behavior

(deviates from requirements)

requirements:
for input i,
give output 2*(i^3)

(so 6 yields 432)

software:

```
i=input(STDIN);
i=double(i);
i=power(i,3);
output(STDOUT,i);
```

error

output (verbose):

```
input: 6
doubling input..
computing power..
output: 1728
```

fault

fault + failure

What is...

testing:

by experiment,

- find errors in software (Myers, 1979)
- establish quality of software (Hetzel, 1988)

a successful test:

- finds at least one error
- gives quality judgment with maximum confidence with minimum effort

Possible Goals of Testing

- Find faults
  - Glenford Myers, *The Art of Software Testing*
- Provide confidence
  - of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults

Testing and Debugging

- **Testing** is the process of determining if a program has any errors.

- When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as **debugging**.

Test Plan

- A test cycle is often guided by a **test plan**.
- A **test case co**nsists of test data to be input to the program and the expected output.
- The **test data** is a set of values, one for each input variable.
- A **test set** is a collection of zero or more test cases.

Correctness versus reliability

- **Correctness** of a program is desirable, it is almost never the objective of testing.
- To establish correctness via testing: test a program with all possible inputs– this is impossible to accomplish.
- Completeness of testing does not necessarily demonstrate that a program is error free.

- **Reliability:** Probability of failure free operation of software over a given time interval and under given conditions. It is the probability of failure free operation of software in its intended environment.

Testing and Verification

- **Program verification** aims at proving the correctness of programs by showing that it contains no errors. This is very different from testing that aims at uncovering errors in a program.

- Program verification and testing are best considered as complementary techniques. In practice, program verification is often avoided, and the focus is on testing

Testing Principles

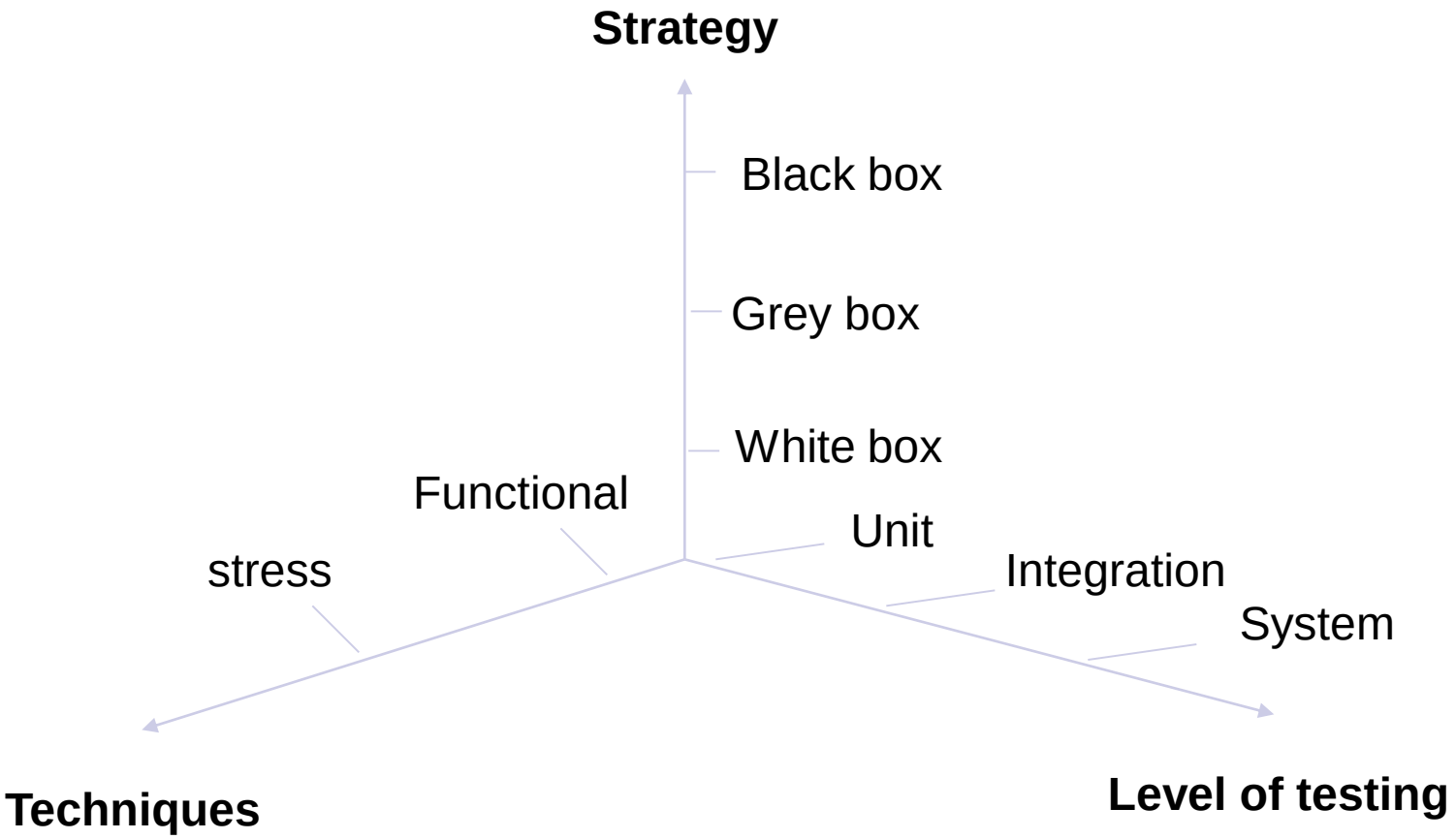
- A Test case must include the definition of expected results
- A programmer should not test her/his own code
- A programming organization should not test its own programs
- Results of each test should be thoroughly inspected
- Test cases must be written for invalid inputs also
- Check that programs do not show unexpected behavior
- Do not plan testing assuming that there are no errors
- The probability of error in a piece of code is proportional to the errors found so far in this part of the code

Standard Testing Questions

- Did this test execution succeed or fail?
  - Oracles
- How shall we select test cases?
  - Selection; generation
- How do we know when we've tested enough?
  - Adequacy
- What do we know when we're done?
  - Assessment

Thumb rule for Testing

- Testing is based on 3 major factors



Test generation strategies

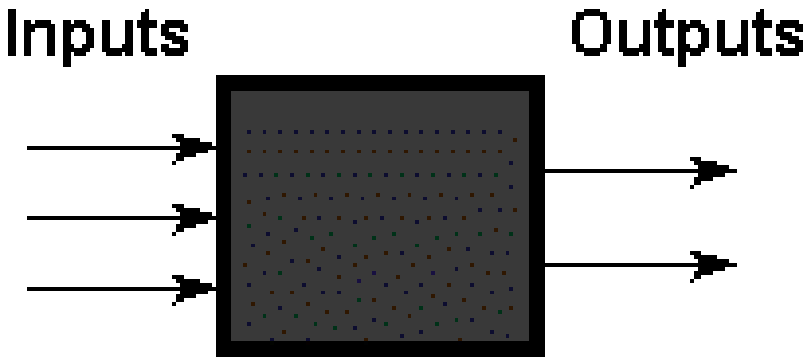
- **Model based:** require that a subset of the requirements be modeled using a formal notation (usually graphical).
  - **Models:** Finite State Machines, Timed automata, Petri net, etc.

- **Specification based:** require that a subset of the requirements be modeled using a formal mathematical notation. Examples: B, Z, and Larch.

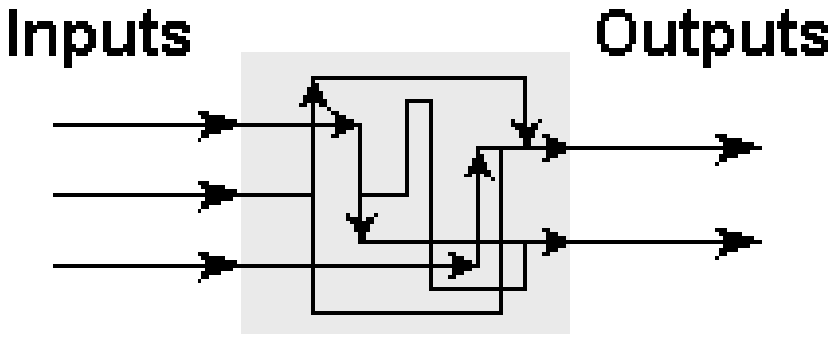
- **Code based:** generate tests directly from the code.

Strategies of testing

Black box



White box



Black and White Box Testing Methods

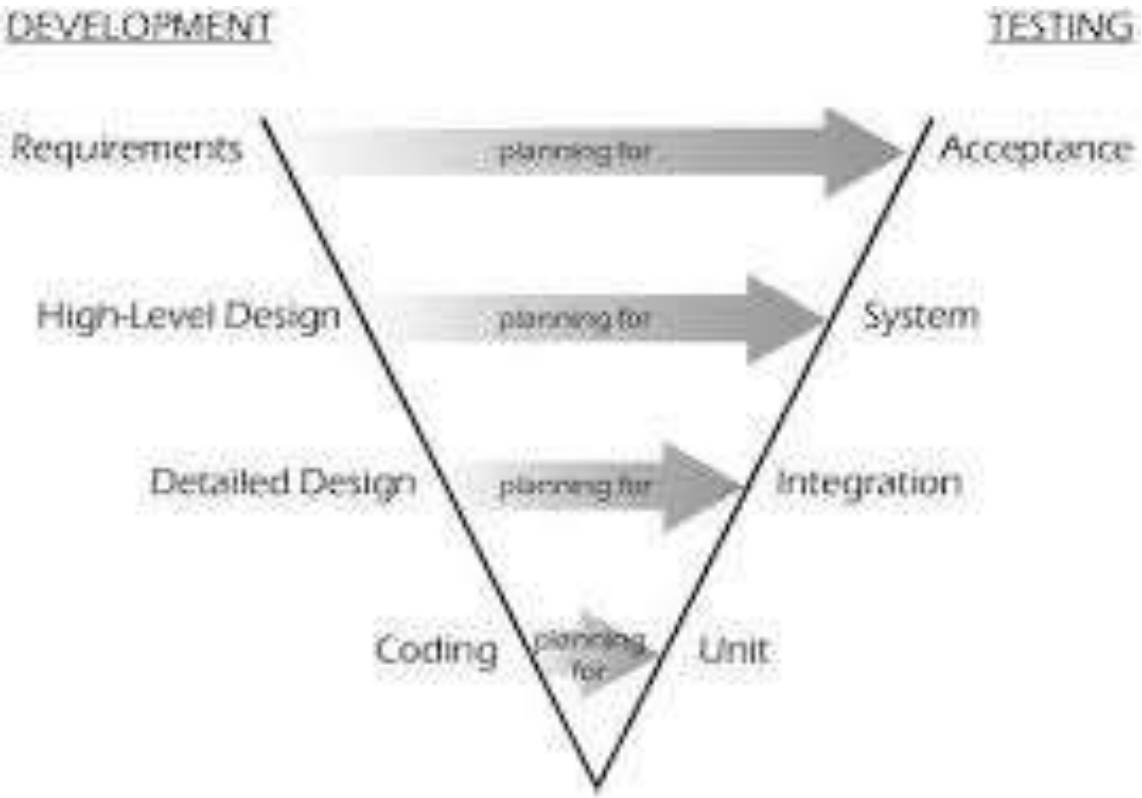
Black box

- Equivalence partitioning
- Boundary-value analysis
- Cause-effect graphing
- Error guessing
- State space exploration

White box

- Coverage/Adequacy
- Data flow analysis
- Cleanness
- Correctness
- Mutation
- Slicing

Levels of abstraction and testing in waterfall model



Ten dominating techniques

- **Function testing**
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

Test each feature or function on its own.

Scan through the product, covering every feature or function with at least enough testing to determine what it does and whether it is working.

Ten dominating techniques

- Function testing
- **Specification-based testing**
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

Check every claim made in the reference document (such as, a contract specification).

Test to the extent that you have proved the claim true or false.

Ten dominating techniques

- Function testing
- Specification-based testing
- **Domain testing**
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

Focus on variables, such as inputs, outputs, configuration, or internal (e.g. file-handling) variables.

For every variable or combination of variables, consider the space of possible values. Simplify it by partitioning into subsets. Pick a few representatives of each subset.

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- **Risk-based testing**
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

A program is a collection of opportunities for things to go wrong.

For each way that you can imagine the program failing, design tests to determine whether the program actually will fail in that way.

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- **Scenario testing**
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

Tests are complex stories that capture how the program will be used in real-life situations.

These are combination tests, whose combinations are credible reflections of real use.

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- **Regression testing**
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

Repeat the same test after some change to the program.

You can use any test as a regression test, but if you do a lot of regression testing, you will (or should) learn to design cases for efficient reuse.

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- **Stress testing**
- User testing
- State-model based testing
- High volume automated testing

Many definitions of stress testing.

When I say stress testing, I mean tests intended to overwhelm the product, to subject it to so much input, so little memory, such odd combinations that I expect it to fail and am exploring its behavior as (and after) it fails.

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- **User testing**
- State-model based testing
- High volume automated testing

Give the program to "a user," see what he does with it and how it responds.

User tests can be tightly structured or very loosely defined. The essence is room for action and response by "users".

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- **State-model based testing**
- High volume automated testing

Model the program as a state machine that runs from state to state in response to events (such as new inputs).

In each state, does it respond correctly to each event?

Ten dominating techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- **High volume automated testing**

Program the computer to design, implement, execute and interpret a large series of tests.

You set the wheel in motion, supply the oracle(s) and evaluate the pattern of results.

Coming to Grips With Reality

- Software will be shipped with bugs
  - Known and unknown issues
  - Due to time constraints, lack of test coverage
- Users will do things the designers never imagined or intended
  - Either accidentally or cleverly
  - With benign and malicious intent
- Plan to fix bugs after a release
  - Software is never “done”
  - Must plan ahead for releasing updates

Generalized Pseudocode

- Pseudocode provides a *language-neutral* way to express program source code.
- Language element – Generalised Pseudocode examples:
  - Comment - ‘<text>
  - Data Structure declaration - Type <type name>
    <list of field descriptions>
    End <type name>
  - Assignment statement - <variable> = <expression>
  - Input - Input (<variable list>)
  - Output - Output (<variable list>)

- The triangle problem
- The NextDate function
- The commission problem
- The SATM (Simple Automatic Teller Machine) problem
- The currency converter problem
- Saturn wind shield wiper problem

Chapter 1:

Preliminaries: Software Testing

Learning Objectives

- Errors, Testing, debugging, test process, CFG, correctness, reliability, oracles.
- Finite state machines
- Testing techniques

1.1 Humans, errors and testing

Errors

Errors are a part of our daily life.

Humans make errors in their thoughts, actions, and in the products that might result from their actions.

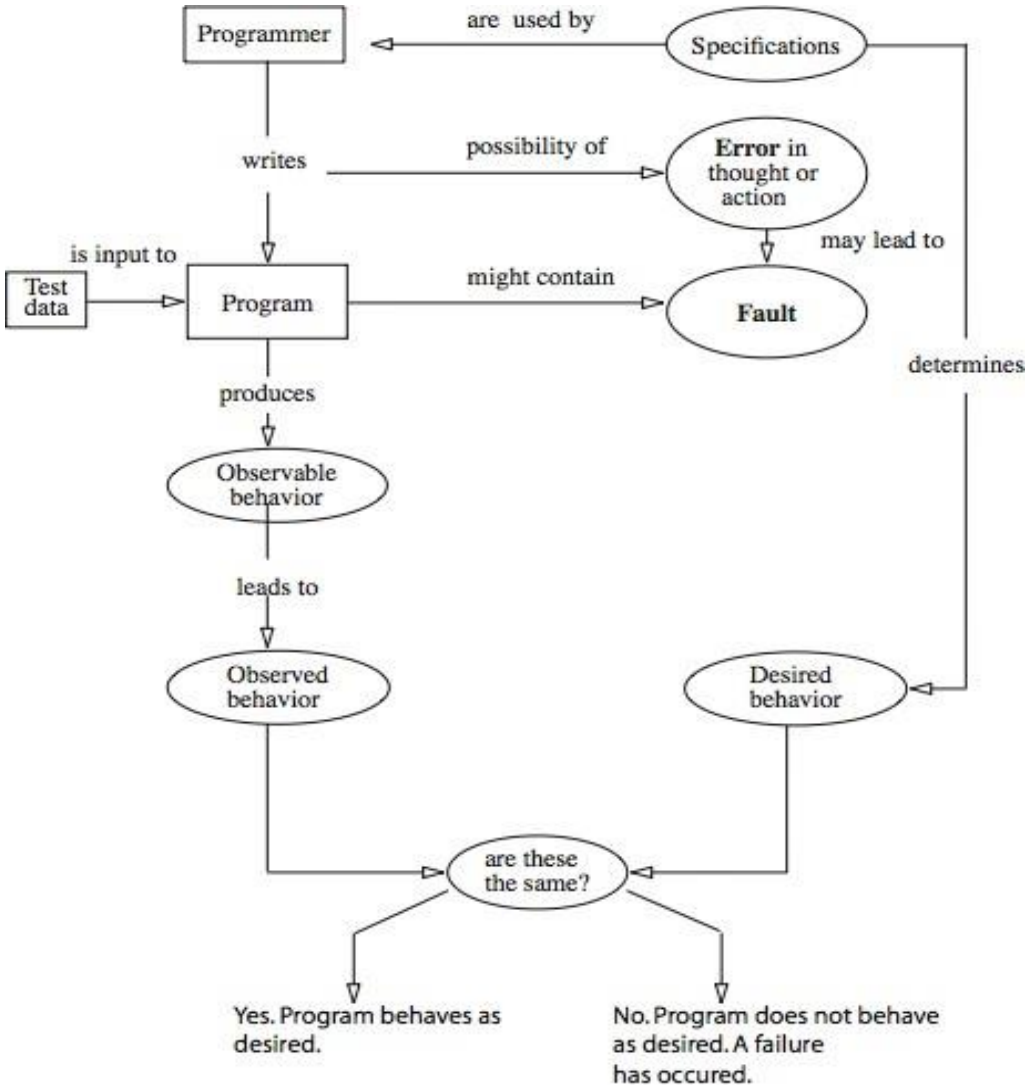
Errors occur wherever humans are involved in taking actions and making decisions.

These fundamental facts of human existence make testing an essential activity.

Errors: Examples

Area	Error
Hearing	Spoken: He has a garage for repairing *foreign* cars.
	Heard: He has a garage for repairing *falling* cars.
Medicine	Incorrect antibiotic prescribed.
Music performance	Incorrect note played.
Numerical analysis	Incorrect algorithm for matrix inversion.
Observation	Operator fails to recognize that a relief valve is stuck open.
Software	Operator used: $\neq$, correct operator: $>$.
	Identifier used: new_line, correct identifier: next_line.
	Expression used: $a \wedge (b \vee c)$, correct expression: $(a \wedge b) \vee c$.
	Data conversion from 64-bit floating point to 16-bit integer not protected (resulting in a software exception).
Speech	Spoken: *waple malnut*, intent: *maple walnut*.
	Spoken: *We need a new refrigerator*, intent: *We need a new washing machine*.
Sports	Incorrect call by the referee in a tennis match.
Writing	Written: What kind of *pans* did you use?
	Intent: What kind of *pants* did you use?

Error, faults, failures



1.2 Software Quality

Software quality

Types of attributes:

- Static
- Dynamic

Static quality attributes: structured, maintainable, testable code as well as the availability of correct and complete documentation.

Dynamic quality attributes: software reliability, correctness, completeness, consistency, usability and performance

Software quality (contd.)

Completeness refers to the availability of all features listed in the requirements, or in the user manual. An incomplete software is one that does not fully implement all features required.

Consistency refers to adherence to a common set of conventions and assumptions. For example, all buttons in the user interface might follow a common color coding convention. An example of inconsistency would be when a database application displays the date of birth of a person in the database.

Software quality (contd.)

Usability refers to the ease with which an application can be used. This is an area in itself and there exist techniques for usability testing. Psychology plays an important role in the design of techniques for usability testing.

Performance refers to the time the application takes to perform a requested task. It is considered as a *non-functional requirement*. It is specified in terms such as ``This task must be performed at the rate of X units of activity in one second on a machine running at speed Y, having Z gigabytes of memory."

1.3 Requirements, behavior, and correctness

Contents

Requirements, behavior, correctness

Requirements leading to two different programs:

Requirement 1: It is required to write a program that inputs two integers and outputs the maximum of these.

Requirement 2: It is required to write a program that inputs a sequence of integers and outputs the sorted version of this sequence.

Contents

Requirements: Incompleteness

Suppose that program **max** is developed to satisfy Requirement 1. The expected output of **max** when the input integers are 13 and 19 can be easily determined to be 19.

Suppose now that the tester wants to know if the two integers are to be input to the program on one line followed by a carriage return, or on two separate lines with a carriage return typed in after each number. The requirement as stated above fails to provide an answer to this question.

Contents

Requirements: Ambiguity

Requirement 2 is ambiguous. It is not clear whether the input sequence is to be sorted in ascending or in descending order. The behavior of **sort** program, written to satisfy this requirement, will depend on the decision taken by the programmer while writing **sort**.

Contents

Input domain (Input space)

*The set of all possible inputs to a program **P** is known as the input domain or input space, of **P**.*

Using Requirement 1 above we find the input domain of **max** to be the set of all pairs of integers where each element in the pair integers is in the range -32,768 till 32,767.

Using Requirement 2 it is not possible to find the input domain for the sort program.

Contents

Input domain (Continued)

Modified Requirement 2:

It is required to write a program that inputs a

sequence of integers and outputs the integers in this sequence sorted in either ascending or descending order. The order of the output sequence is determined by an input request character which should be ``A'' when an ascending sequence is desired, and ``D'' otherwise.

While providing input to the program, the request character is input first followed by the sequence of integers to be sorted; the sequence is terminated with a period.

Contents

Input domain (Continued)

Based on the above modified requirement, the input domain for **sort** is a set of pairs. The first element of the pair is a character. The second element of the pair is a sequence of zero or more integers ending with a period.

Contents

Valid/Invalid Inputs

The modified requirement for sort mentions that the request characters can be ``A" and ``D", but fails to answer the question ``What if the user types a different character ?’’

When using **sort** it is certainly possible for the user to type a character other than ``A" and ``D". Any character other than ``A"’ and ``D"is considered as invalid input to **sort**. The requirement for **sort** does not specify what action it should take when an invalid input is encountered.

Contents

1.4 Correctness versus reliability

Contents

Correctness

Though correctness of a program is desirable, it is almost never the objective of testing.

To establish correctness via testing would imply testing a program on all elements in the input domain. In most cases that are encountered in practice, this is impossible to accomplish.

Thus, correctness is established via mathematical proofs of programs.

Contents

Correctness and Testing

While correctness attempts to establish that the program is error free, testing attempts to find if there are any errors in it.

Thus, completeness of testing does not necessarily demonstrate that a program is error free.

Testing, debugging, and the error removal processes together increase our confidence in the correct functioning of the program under test.

Contents

Software reliability: two definitions

Software reliability [ANSI/IEEE Std 729-1983]: is the probability of failure free operation of software over a given time interval and under given conditions.

Software reliability is the probability of failure free operation of software in its intended environment.

Contents

Operational profile

An operational profile is a numerical description of how a program is used.

Consider a sort program which, on any given execution, allows any one of two types of input sequences. Sample operational profiles for sort follow.

Contents

Operational profile

Operational profile #1

Sequence	Probability
Numbers only	0.9
Alphanumeric strings	0.1

Contents

Operational profile

Operational profile #2

Sequence	Probability
Numbers only	0.1
Alphanumeric strings	0.9

Contents

1.5 Testing and debugging

Contents

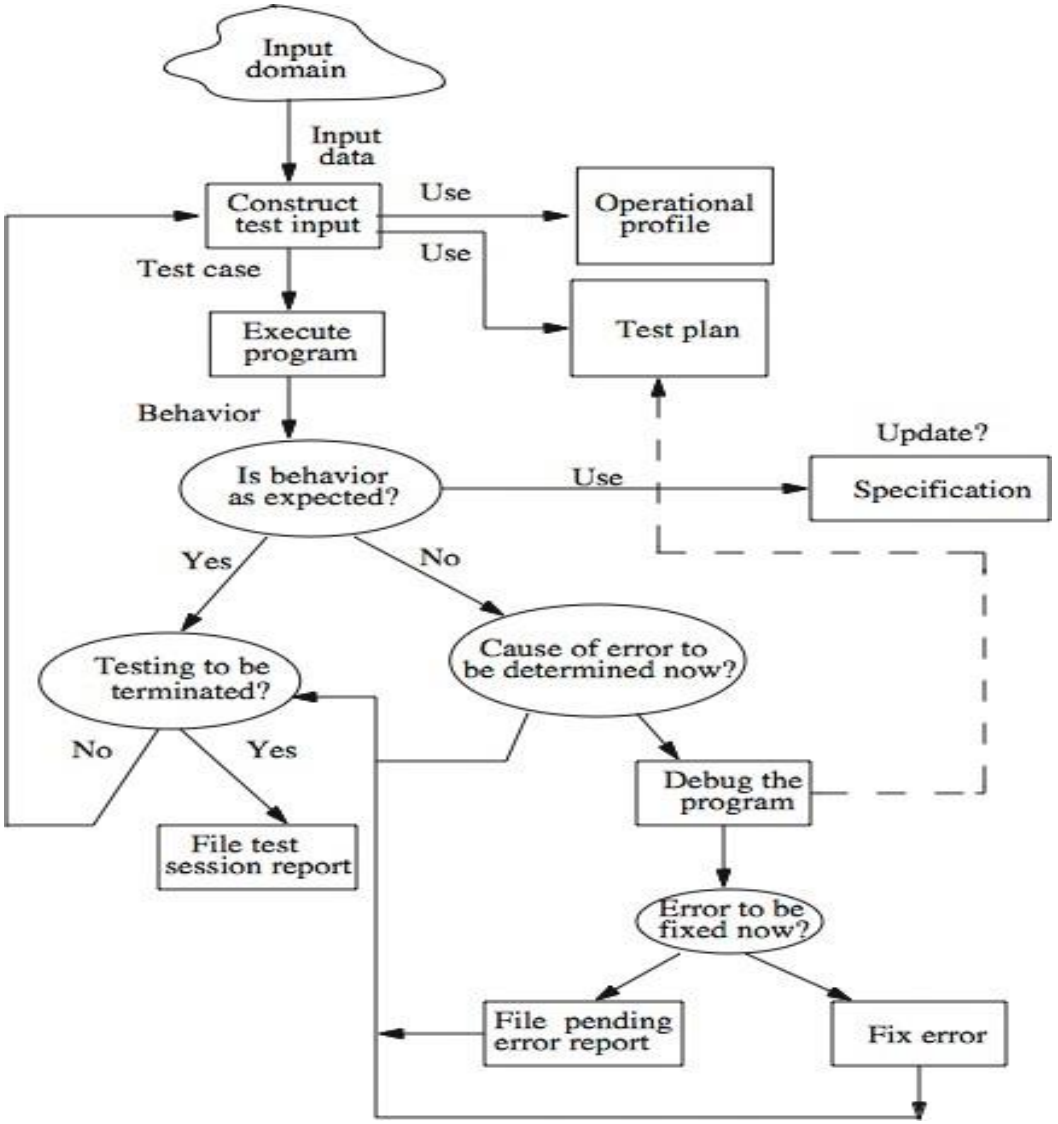
Testing and debugging

Testing is the process of determining if a program has any errors.

When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as debugging.

Contents

A test/debug cycle



Contents

Test plan

A test cycle is often guided by a test plan.

Example: The sort program is to be tested to meet the requirements given earlier. Specifically, the following needs to be done.

- Execute sort on at least two input sequences, one with ``A'' and the other with ``D'' as request characters.

Contents

Test plan (contd.)

- Execute the program on an empty input sequence.

- Test the program for robustness against erroneous inputs such as ``R'' typed in as the request character.

- All failures of the test program should be recorded in a suitable file using the Company Failure Report Form.

Contents

Test case/data

A test case is a pair consisting of test data to be input to the program and the expected output. The test data is a set of values, one for each input variable.

A test set is a collection of zero or more test cases.

Sample test case for sort:

Test data: <"A' 12 -29 32 >

Expected output: -29 12 32

Contents

Program behavior

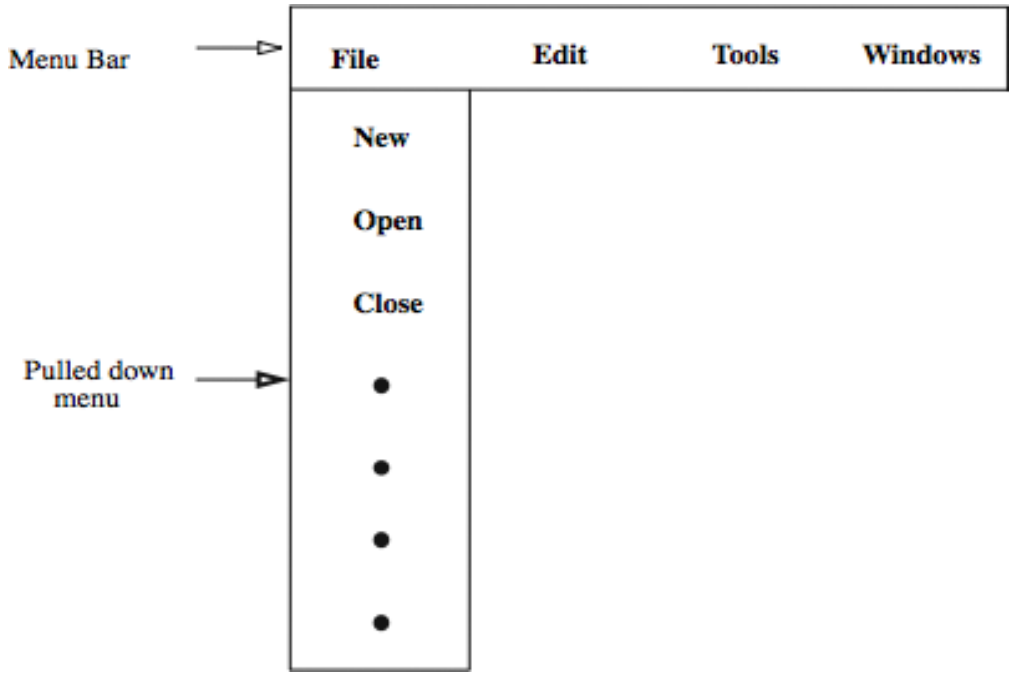
Can be specified in several ways: plain natural language, a state diagram, formal mathematical specification, etc.

A state diagram specifies program states and how the program changes its state on an input sequence. inputs.

Contents

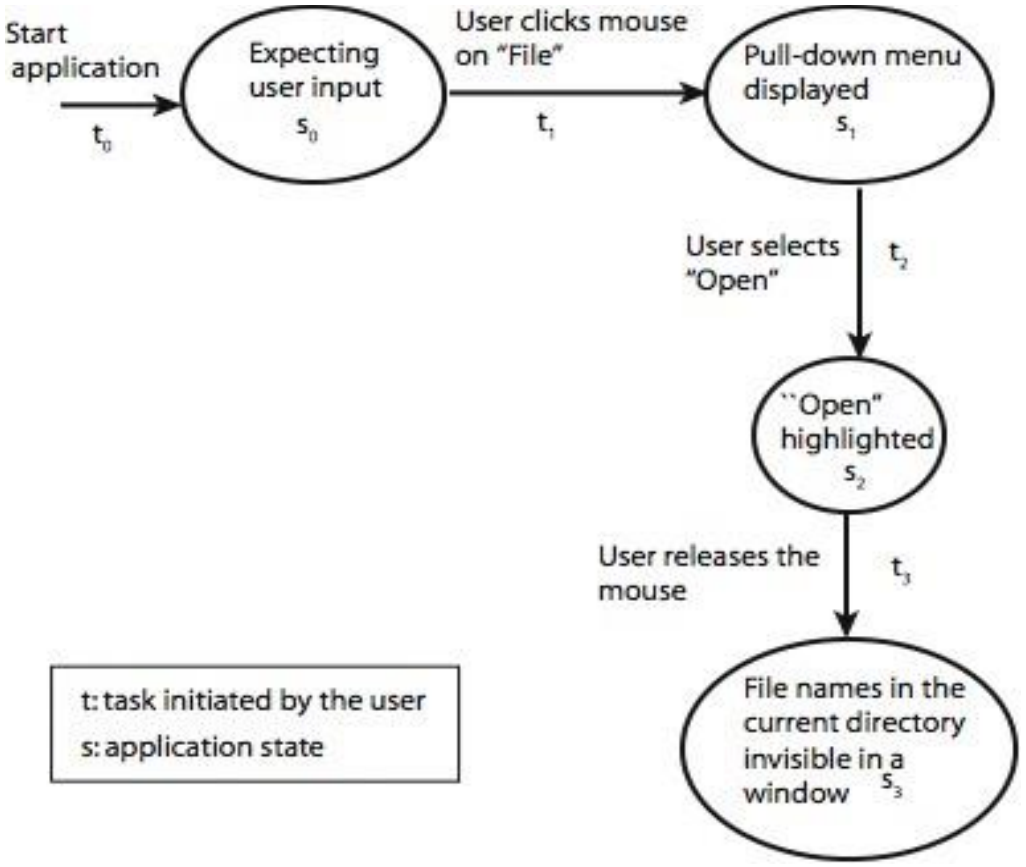
Program behavior: Example

Consider a menu driven application.



Contents

Program behavior: Example (contd.)



Contents

Behavior: observation and analysis

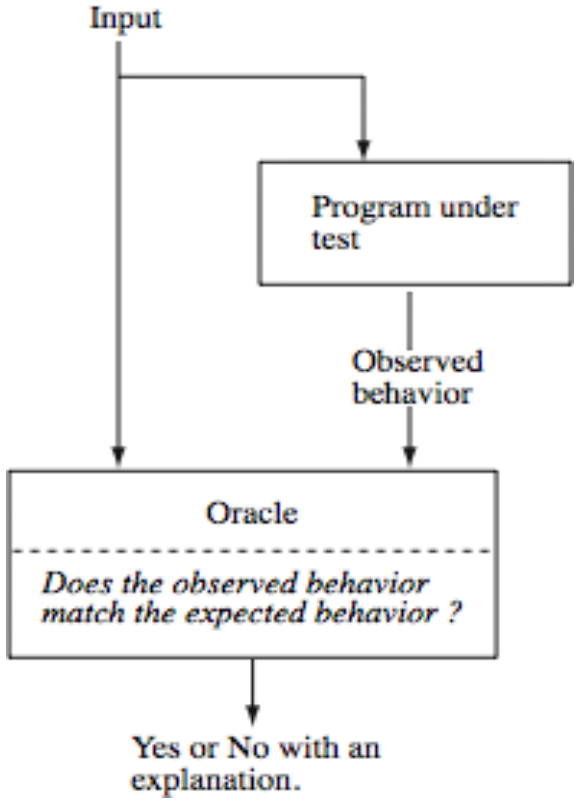
In the first step one observes the behavior.

In the second step one analyzes the observed behavior to check if it is correct or not. Both these steps could be quite complex for large commercial programs.

The entity that performs the task of checking the correctness of the observed behavior is known as an oracle.

Contents

Oracle: Example



Contents

Oracle: Programs

Oracles can also be programs designed to check the behavior of other programs.

For example, one might use a matrix multiplication program to check if a matrix inversion program has produced the correct output. In this case, the matrix inversion program inverts a given matrix A and generates B as the output matrix.

Contents

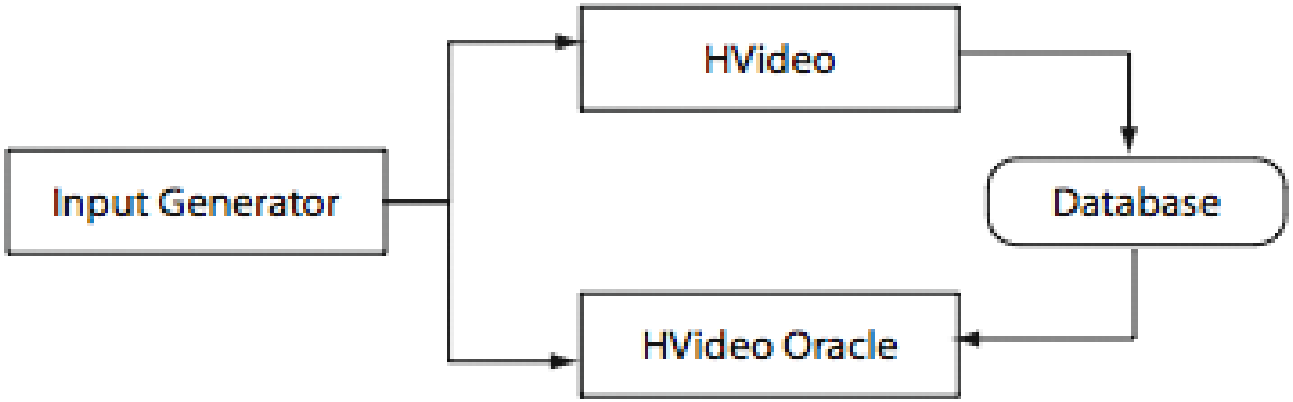
Oracle: Construction

Construction of automated oracles, such as the one to check a matrix multiplication program or a sort program, requires the determination of input-output relationship.

In general, the construction of automated oracles is a complex undertaking.

Contents

Oracle construction: Example



Contents

Testing and verification

Program verification aims at proving the correctness of programs by showing that it contains no errors. This is very different from testing that aims at uncovering errors in a program.

Program verification and testing are best considered as complementary techniques. In practice, program verification is often avoided, and the focus is on testing.

Contents

Testing and verification (contd.)

Testing is not a perfect technique in that a program might contain errors despite the success of a set of tests.

Verification promises to verify that a program is free from errors. However, the person/tool who verified a program might have made a mistake in the verification process; there might be an incorrect assumption on the input conditions; incorrect assumptions might be made regarding the components that interface with the program, and so on.

Verified and published programs have been shown to be incorrect.

Contents

1.10. Test generation strategies

Contents

Test generation

Any form of test generation uses a source document. In the most informal of test methods, the source document resides in the mind of the tester who generates tests based on a knowledge of the requirements.

In several commercial environments, the process is a bit more formal. The tests are generated using a mix of formal and informal methods either directly from the requirements document serving as the source. In more advanced test processes, requirements serve as a source for the development of formal models.

Contents

Test generation strategies

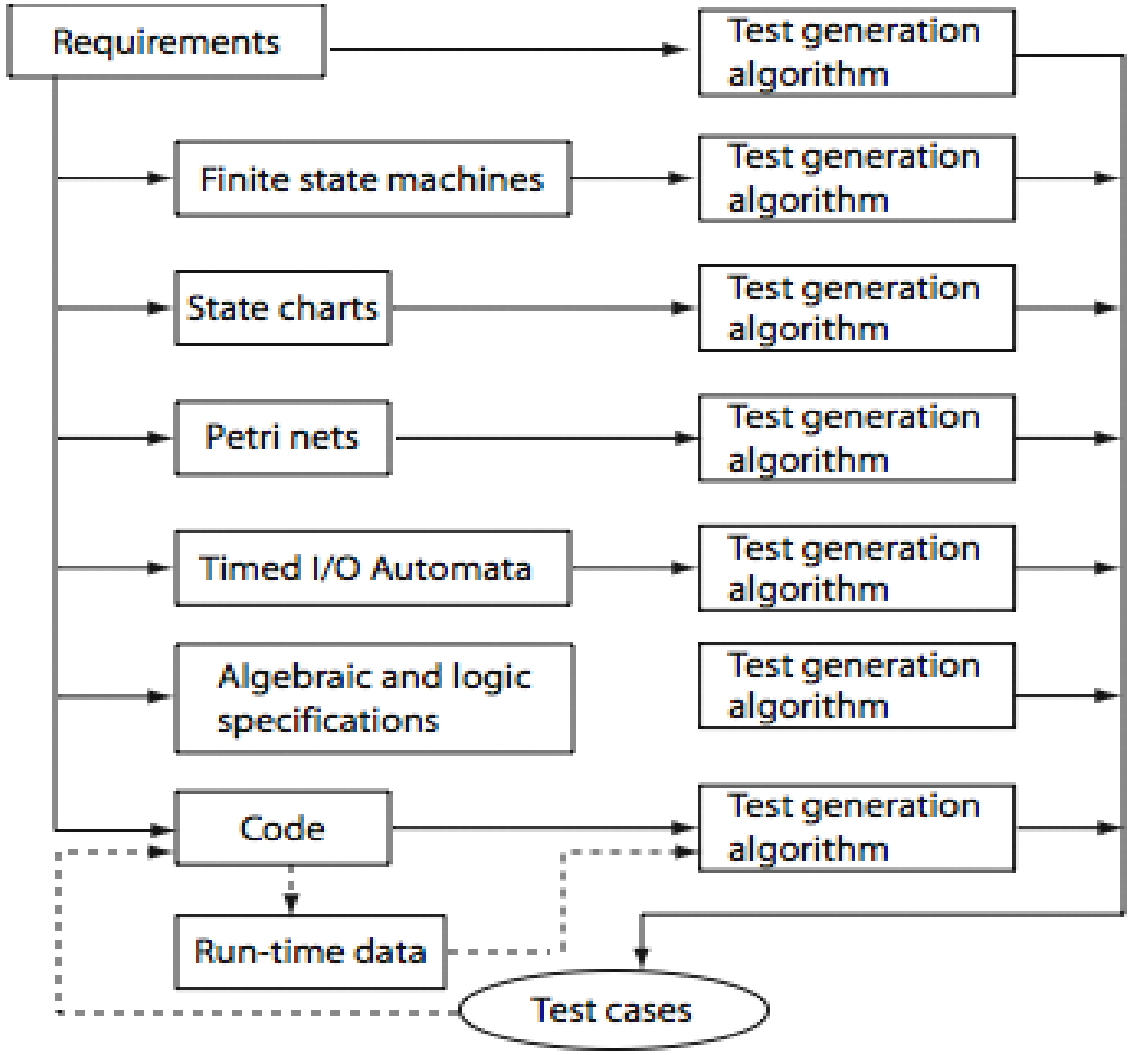
Model based: require that a subset of the requirements be modeled using a formal notation (usually graphical). Models: Finite State Machines, Timed automata, Petri net, etc.

Specification based: require that a subset of the requirements be modeled using a formal mathematical notation. Examples: B, Z, and Larch.

Code based: generate tests directly from the code.

Contents

Test generation strategies (Summary)



Contents

1.13 Types of software testing

Contents

Types of testing

One possible classification is based on the following four classifiers:

C1: Source of test generation.

C2: Lifecycle phase in which testing takes place

C3: Goal of a specific testing activity

C4: Characteristics of the artifact under test

Contents

C1: Source of test generation

Artifact	Technique	Example
Requirements (informal)	Black-box	Ad-hoc testing Boundary value analysis Category partition Classification trees Cause-effect graphs Equivalence partitioning Partition testing Predicate testing Random testing
Code	White-box	Adequacy assessment Coverage testing Data-flow testing Domain testing Mutation testing Path testing Structural testing Test minimization using coverage
Requirements and code	Black-box and White-box	
Formal model: Graphical or mathematical specification	Model-based Specification	Statechart testing FSM testing Pairwise testing Syntax testing
Component interface	Interface testing	Interface mutation Pairwise testing

Contents

C2: Lifecycle phase

Phase	Technique
Coding	Unit testing
Integration	Integration testing
System integration	System testing
Maintenance	Regression testing
Post system, pre-release	Beta-testing

Contents

C3: Goal of specific testing activity

Goal	Technique	Example
Advertised features	Functional testing	
Security	Security testing	
Invalid inputs	Robustness testing	
Vulnerabilities	Vulnerability testing	
Errors in GUI	GUI testing	Capture/plaback
		Event sequence graphs
		Complete Interaction Sequence
Operational correctness	Operational testing	Transactional-flow
Reliability assessment	Reliability testing	
Resistance to penetration	Penetration testing	
System performance	Performance testing	Stress testing
Customer acceptability	Acceptance testing	
Business compatibility	Compatibility testing	Interface testing
		Installation testing
Peripherals compatibility	Configuration testing	

Contents

C4: Artifact under test

Characteristics	Technique
Application component	Component testing
Client and server	Client-server testing
Compiler	Compiler testing
Design	Design testing
Code	Code testing
Database system	Transaction-flow testing
OO software	OO testing
Operating system	Operating system testing
Real-time software	Real-time testing
Requirements	Requirement testing
Software	Software testing
Web service	Web service testing

Contents

Summary

We have dealt with some of the most basic concepts in software testing. Exercises at the end of Chapter 1 are to help you sharpen your understanding

Contents

SOFTWARE TESTING

To err is human

Since committing errors is an inevitable component in all facets of any application development, it is highly required to detect and eliminate the errors. In software projects, error injection is a common act which can get seeded either in the product or during the process of developing the product. Hence, such errors should be identified and eliminated. There is always difference between error, fault, failure and defect.

IEEE definitions for the above are as below:

Error: Human mistake that caused fault
Fault: Discrepancy in code that causes a failure.
Failure: External behavior is incorrect
Defect: Fault when become visible can act as a defect.

There are two ways in which errors can be addressed namely quality control and quality assurance.

Quality Control – The process by which product quality is compared and detected with respect to requirements and other relevant specifications, focus is in detection and removal. Testing is one activity through which quality control can be achieved.

Quality Assurance – The set of activities (including facilitation, training, measurement, and analysis) needed to provide adequate confidence that process are established continuously improved to produce products that meet specifications and are fit for use. There are several techniques through which quality assurance can be achieved namely reviews, walkthroughs, inspection, training, audit assessment, metrics and standards through which process can be improved.

Software Testing: A Quality control activity aimed at evaluating a software item against the given system requirements

Some definitions of testing:

- Definition (1)
  - Process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirement (IEEE 83a)
- Definition (2)
  - Structured process that uncovers the defects in a software product.
  - Destructive in nature (dismantling the wishful assumption that code is bug-free)
- Definition (3)
  - Testing is a process of executing a program with the intent of finding errors (Myers).

Significance of testing:

- Ensure product works with negligible risks.
- Find defects in product
- Demonstrate the lack of quality
- Client or End user should not find bugs
- Detect programming errors
- Demonstrate difference between specifications and developed system
- Establish confidence in the product

Limitations of testing:

- Does not generalize system behavior
- Does not guarantee bug free product
- No substitute for good programming

Testing can be classified into various types of testing and various levels of testing

Levels of Testing:

- Unit test
- Integration test
- System test
- User acceptance test

Unit testing is an activity which will be conducted by both testers and developers. It is act where every deliverable which is developed by the developer will be tested and also testers will test it from both functionality check and non functionality checks. Example: Testing the login module of online library management system.

Integration testing is also usually tested by programmers. However, there can be testers who do interface testing. Main issue in this type of testing is the choice of integrating i.e top down integration test or bottom up integration test or sandwich integration test. Example: Testing the login module with book taken module and payment module of the online library management system.

System test is an activity where the developed system after unit tested, integration tested will be once again test the entire system in order to validate if all specifications are met or not. This will be usually tested by testers. Example: Testing complete online library management system.

User acceptance test is the final test subjected to the customers to test the product before it is deployed to them. This test decides to either accept the product or not depending on satisfaction level of all specifications in the product developed. Example: Testing the online library management system by customer.

Types of testing:

- Alpha test
- Beta test
- Black box test
- While box test
- Random test
- Regression test
- Performance test
- Load test
- Stress test
- Exploratory test
- Combinatorial test
- Smoke test

i. Alpha test:

Testing the product by the developers in presence of customers in house of development is alpha test. Example: Testing online library management software in the company who developed and showing them how it operates to customers

ii. Beta test:

Testing the product by customers at their location in absence of development team is beta testing. Example: Testing online library management software in the customer's location by them.

iii. Black box testing:

Testing the product by giving input and obtaining the output is black box test. Example: Giving login and checking if it logs in or not in the online library management system

iv. White box test:

Testing the code either line by line or condition by condition or loop by loop or by path are white box test. Example: tracing every statement of login module of online library management system.

v. Random test:

Random testing involves selecting test cases based on a probability distribution. It is not ad hoc test. Example: Testing the online library management system by giving random number of customers to login and check if they are legitimate users or not

vi. Regression test:

This is the most common test run on any application. Intention is not just to detect defects in the product but also once debugged, the corrected module is once again subjected to test from the beginning and also to test all those modules to which the corrected module has a dependency.

Example: test login module and found a error while navigating to menu page. Having debugged the error, once again retest login module and also all those modules to which this login is associated with. Aim of such test is to check for bad fixes. Bad fix is introducing new defects by debugging exiting defect.

vii. Performance test:

Performance testing is the process of determining the system's performance that includes speed, reliability under varying load. Example: Test the login module by readers to login in large numbers to check if the system goes to down time or does performance decrease. In this case, it is considered to test the login module in normal conditions.

viii. Load test:

Load testing is the process of determination of behavior of system when multiple users access it at the same time. This is once again testing with expected load given to system. Example: test the behavior of login module when 1000 readers login at the same time.

ix. Stress test:

Stress Testing is performed to test the robustness of the system or software application under extreme load. Example: Making 1500 readers to login at the same time and to test to see if the performance is decreased or not.

x. Exploratory test:

Exploratory testing is one such manual testing approach where test cases are generated based on the experience and knowledge of the tester. Test cases in exploratory testing are not pre defined unlike conventional testing but are generated and executed simultaneously. Example: Testers testing login module with already generated test case find possibility of testing with few more creative test cases which are not available in test suite.

xi. Combinatorial test:

Combinatorial testing (CT) which is a black box testing method is carried out on how the variable and their combinations are selected to perform proficient testing with the least possible number of test cases covering maximum critical aspects of the software. Example: Testing online library management system with all possible combinations of inputs and parameters for specific scenarios.

xii. Smoke test:

Testing the critical functionalities of the system is smoke test. Example: Testing payment dues module of online library management system.

Software testing however a life cycle to be has followed which is popularly known as Software Test Life Cycle (STLC).

Once STLC begins, following activities are conducted as part of test life cycle:

- Test Strategy
- Test Plan
- Test Case Design
- Test Case Generation
- Test Execution
- Regression Test
- Test Closure Report
- Retrospection Move

Test Strategy: This activity decides the approach of testing which includes decision on type of testing, rationale for it so on.

Test Plan: This activity ensures one to plan for the strategy incorporated in terms of time, cost, resources required.

Test Case Design: In this activity, testers who are assigned comes out with design of test case which looks as below

Table 6: Sample Test Case template

Test id	Test case description	Steps followed	Input Data	Expected Output	Actual Output	Test case passed/failed	Remarks

Test Case Generation: In this activity, test case is generated where every specification is analyzed from all perfectives of stakeholders and prepared. Collection of test cases is test suite.

Table 7: Sample Test Case Generation for ATM application as an example

Test id	Test case description	Steps followed	Input Data	Expected Output	Actual Output	Test case passed/failed	Remarks
R101	Login	1	Insert Card	Card is valid			
		2	Enter 3 digit numeric PIN	PIN Is valid			

Test Execution: In this activity, testers carry out actual testing where code will be subjected to test for the test cases generated.

Table 8: Test Execution for the ATM Application as an example for login module sample

Test id	Test case description	Steps followed	Input Data	Expected Output	Actual Output	Test case passed/failed	Remarks
R101	Login	1	Insert Card	Card is valid	Valid Card	Passed	
		2	Enter 3 digit numeric PIN	PIN Is valid	PIN Invalid	Failed	Accepted alphanumeric characters as PIN

Regression Test: This is the most common activity that occurs while testing any application of any domain. During this part of STLC, testers once they identify the defect, it will be sent to the authors of the code to rectify the defect. Having obtained the debugged code, the code will be tested from the beginning and also tested for bad fixes in the modules to which this code has a dependency.

Test Closure Report: In this activity, testers will prepare a report on how many defects were identified, severity and impact of each of the defect, time taken to identify the defect, time taken to rectify and resolve the defect by developers, number of bad fixes reported, test effort and efficiency achieved. The report will be kept in the repository for subsequent use and knowledge.

Retrospection Move: Once the testing process is completed and test report is handed over to the project manager or quality head, testers will introspect to analyze the strengths and weakness of the entire STLC process to find out what went well and what did not go well.

Verification and Validation:

Above process of STLC when takes place, the testers carry out two important modes of testing namely verification and validation

Verification:

"Are we building the product right". It is subjective in nature and conducted for every deliverable. Hence, the software should conform to its specification.

Validation:

"Are we building the right product". It is objective in nature and is conducted for the end product or artefacts. Hence, the software should do what the user really requires.

Software Inspection

Software inspection is visual examination of software to find out the static defects. Static defects are the ones which can be detected through our eyes than requiring running the system. It is usually carried out by an external inspector who is not part of the project and can be either from other project within the organization or an external person to the organization. Carrying out inspection ensures reduction of test time as most of the static defects will be detected through inspection and testing will have to detect only the dynamic defects which are visible only at the execution time.

Software inspection can be carried out for every deliverable such as given below

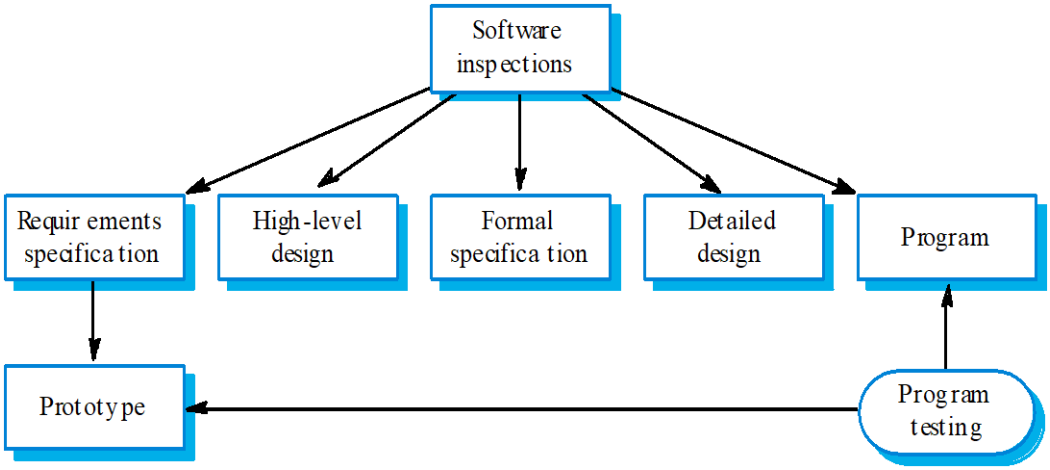


Figure 37: Deliverable upon which software inspection can be conducted.

From the figure, it is clear that software inspection can be carried out on every artifact of every phase of software development.

However, inspection cannot just happen without prior plan and domain knowledge. The figure below shows the preparation phase required to carry out inspection.

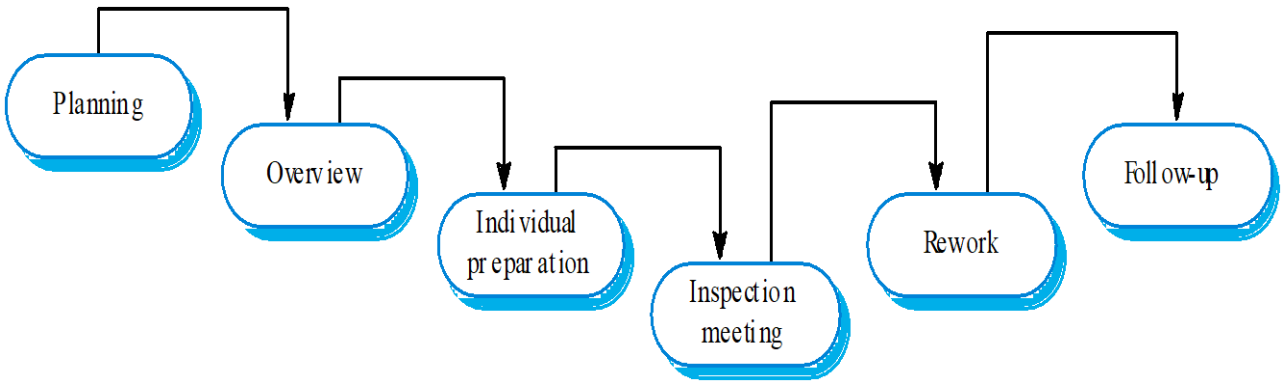


Figure 38: Inspection Process

From the figure, it is clear that inspection will be initially planned for the choice of inspector and their team, time to be given for them, resources to be supported for the inspection process and environment to be set up for the same.

With this plan, an overview of the agenda of the inspection will be prepared to check which modules and which of the deliverables needs to be software inspected. Subsequently, a small amount of time will be provided for the team to prepare themselves for carrying out inspection so that they are aware of the dolman, and other necessary knowledge to do inspection. Actual inspection occurs where defects will be identified and reported. Finally, when defects are fixed and given again for re inspection, if any defects are identified, it will be given for resolving. This process should not be carried out more than two times to ensure effective inspection process.

Thus, the inspection team has their defined roles and responsibilities while doing inspection. This is summarized in the table below

Table 9: Roles and responsibilities of the inspection team members

Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Table above indicates how roles are defined for every member of the inspection team. They will be provided with checklist through which they can verify if the data is defined, if parameters are properly assigned and so on. Deviation of the deliverable from the checklist indicates existence of defects.

While planning the testing, usually V Model is followed as shown below

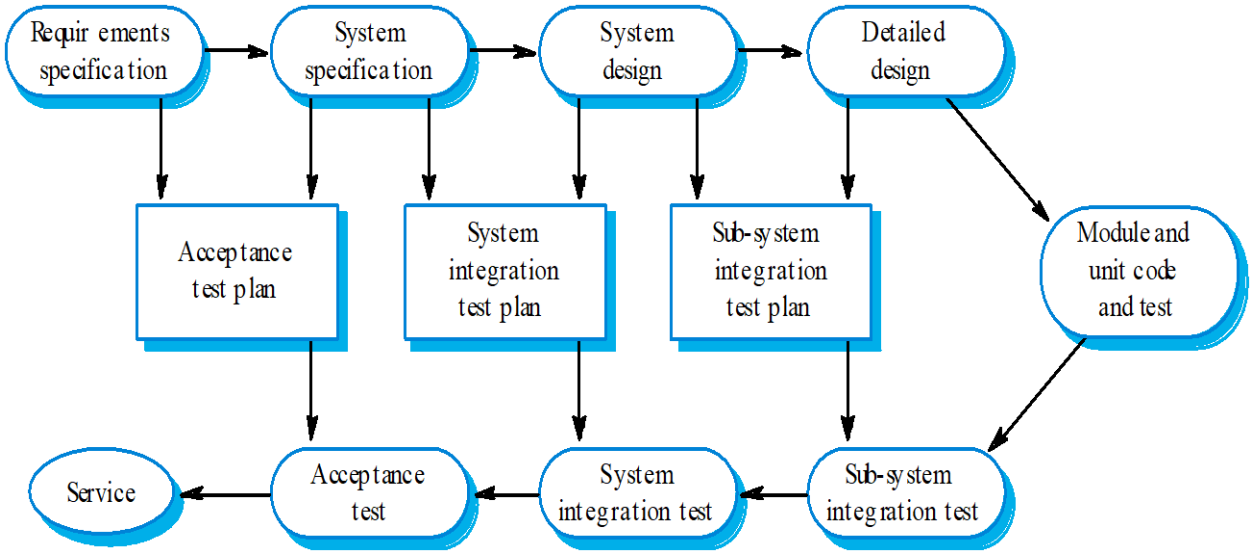


Figure 39: V Model of carrying out Testing

Figure indicates that code is unit tested by developers and testers, once unit code is tested, testers carry out integration testing by integrating the modules and test the subsystem thereby. This mode of testing checks to test the design if design prepared and integrated modules are both matching or has any discrepancy. With sub system testing which tests the design and integration modules along with their interfaces and parameters passing between modules, next step is it to test the entire system. Testers then test system to check if all specifications are met or not. This type of testing is validation testing to see the code maps to specifications. Finally, user acceptance test is conducted to validate if users are satisfied with the system which is developed and if all their requirements are well achieved.

Yet another way of performing testing for critical applications such as software for robotic surgery, space craft to be launched in the space and applications of similar types are tested using clean room approach.

The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal. This approach follows meticulous mode of testing to ensure that the product is clean from all perspectives. This software development process is based on Incremental development, Formal specification, Static verification using correctness arguments and Statistical testing to determine program reliability. Figure below indicates the process of cleanroom testing

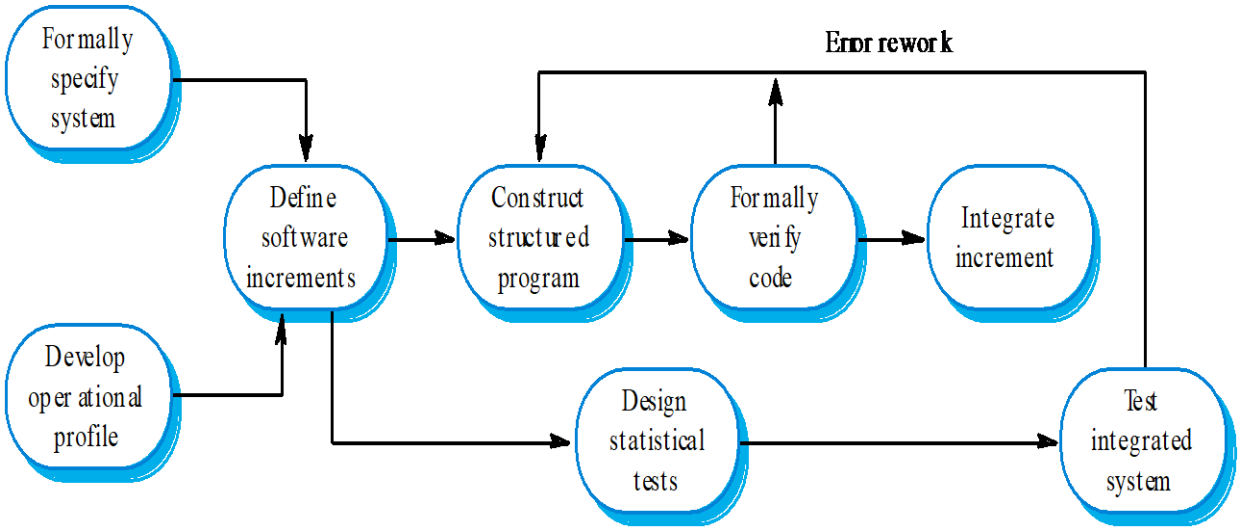


Figure 40: Cleanroom approach of testing

In this approach, requirements are specified in a formal way where mathematically specifications have to be expressed and written. Operational profile in testing indicates a quantitative way of indicating how a system is used by writing the inputs (profile) and anticipating the type of outputs accordingly. With the inputs for the formally written down specifications, every increment of the application will be identified and programs will be written to implement the same. The implemented code will be verified unit wise formally, integrate them and test it in a statistical way using statistical testing such as reliability growth models or any other tests. The final system if satisfied will be then delivered to the customers. This mode of testing in a formal way using statistical testing approach for the formally specified requirements ensures maximum defect free products.

This approach is carried out by a set of team members who include:

- Specification team: who are responsible for developing and maintaining the system specifications
- Development team: who is responsible for developing and verifying the software and the software is NOT executed or even compiled during this process
- Certification team: who are responsible for developing a set of statistical tests to exercise the software after development and generally Reliability growth models are used to determine when reliability is acceptable.