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.

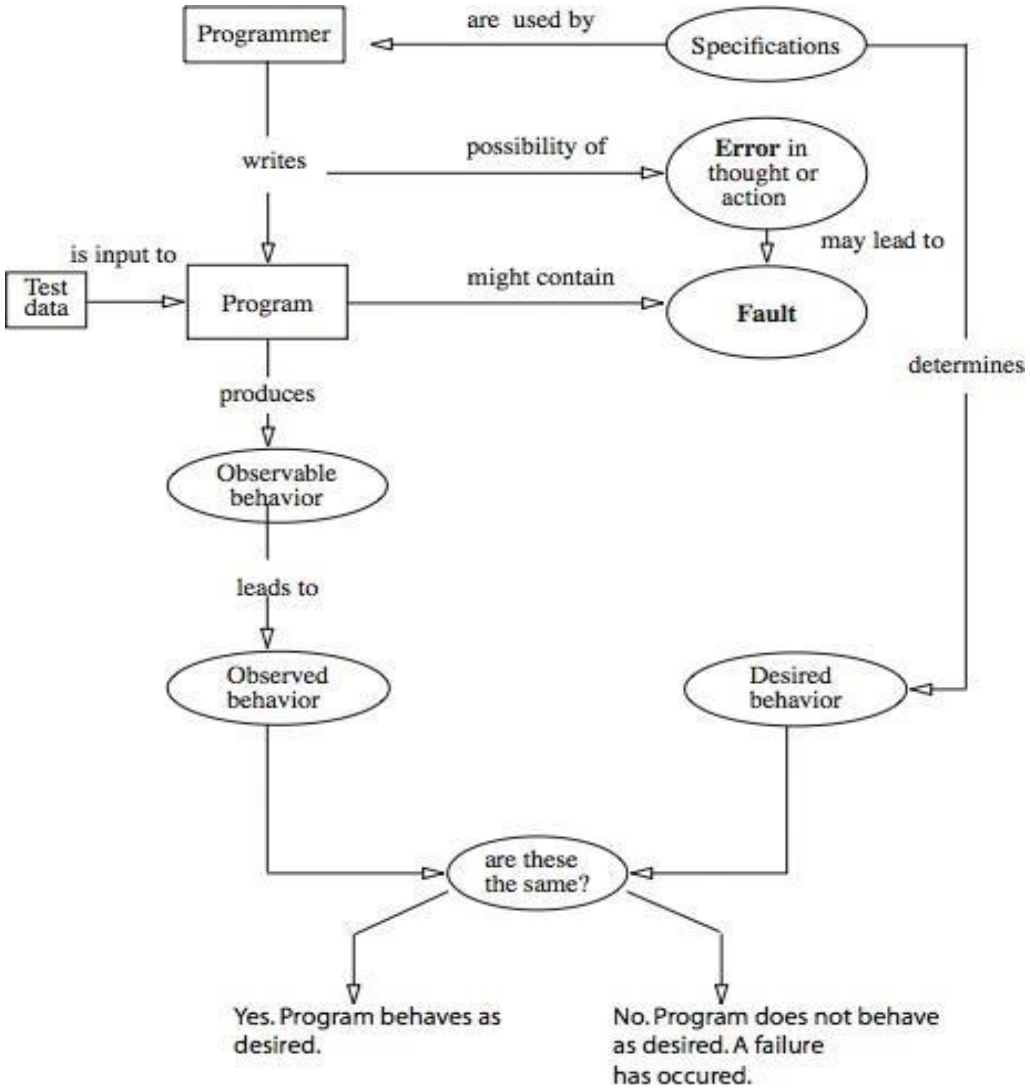
Errors occur wherever humans are involved in taking actions and making decisions.

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



1.2Software 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

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.

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.

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**.

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.

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.

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.

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.

1.4 Correctness versus reliability

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.

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.

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.

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.

Operational profile

Operational profile #1

Sequence	Probability
Numbers only	0.9
Alphanumeric strings	0.1

Operational profile

Operational profile #2

Sequence	Probability
Numbers only	0.1
Alphanumeric strings	0.9

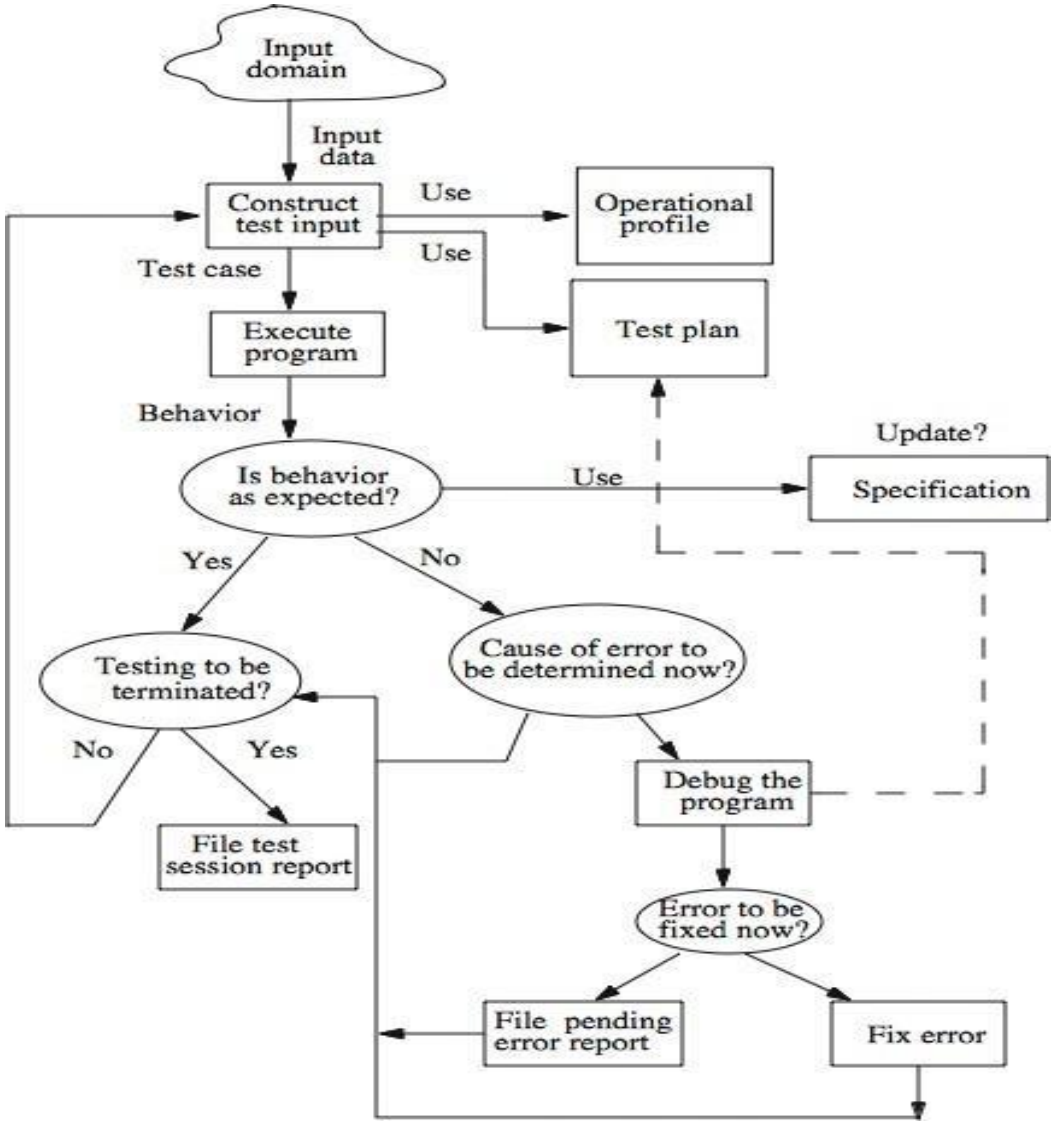
1.5Testing and debugging

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.

A test/debug cycle



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.

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.

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

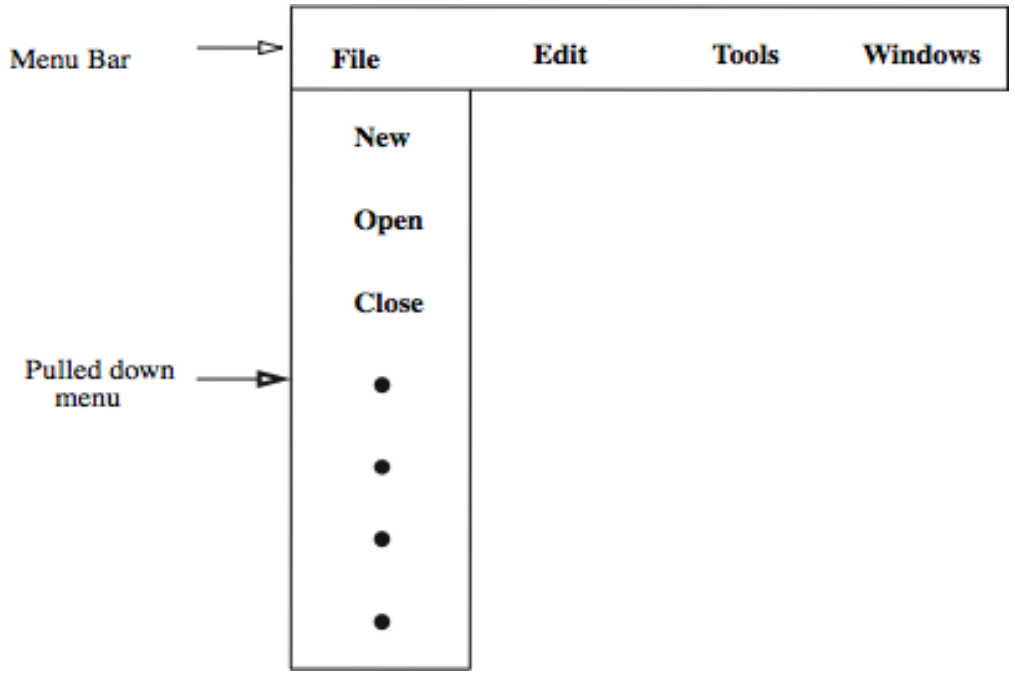
Program behavior

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

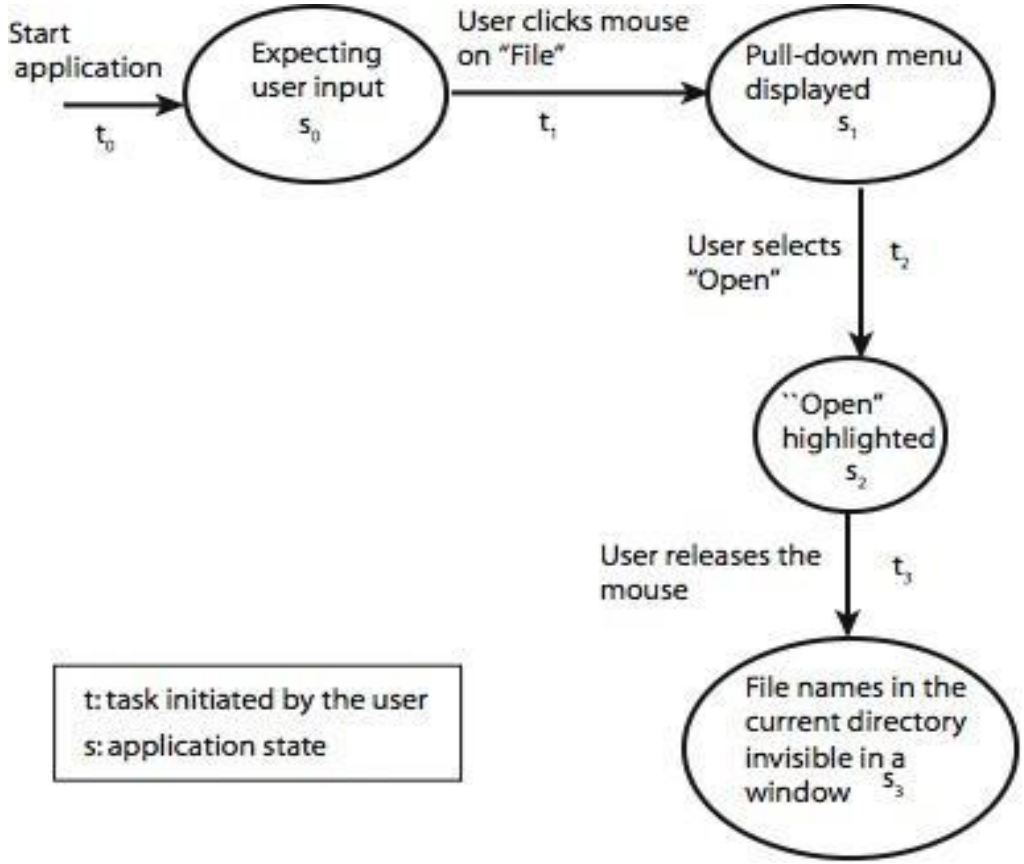
A state diagram specifies program states and how the program changes its state on an input sequence. inputs.

Program behavior: Example

Consider a menu driven application.



Program behavior: Example (contd.)



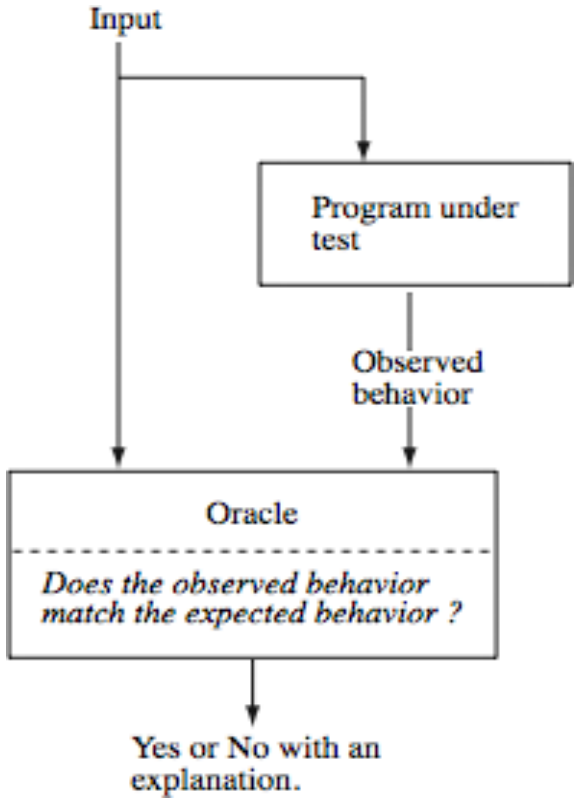
Behavior: observation and analysis

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.

Oracle: Example



Oracle: Programs

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

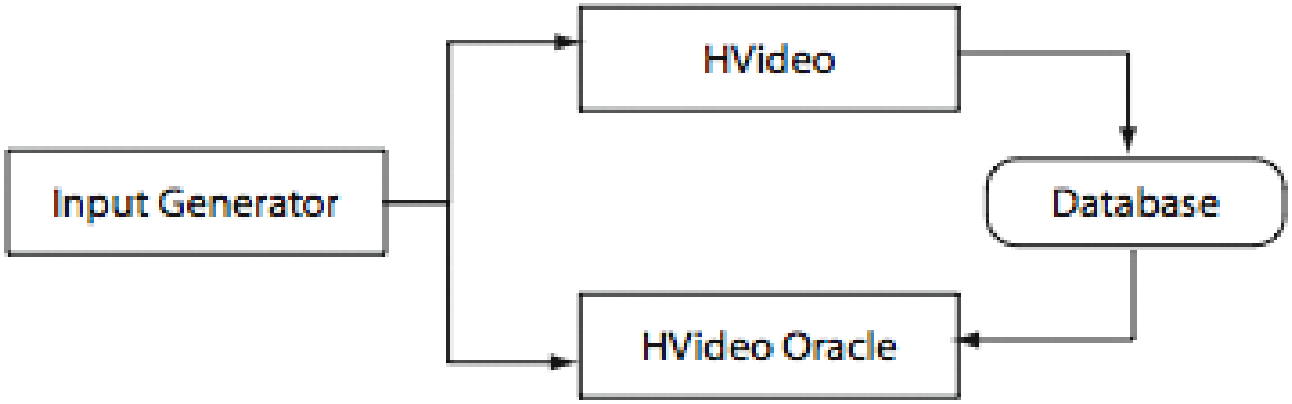
For example, one might use a matrix multiplication program to check if a matrix inversion program has produced the correct output. In this case, the matrix inversion program inverts a given matrix A and generates B as the output matrix.

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.

Oracle construction: Example



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 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.

1.10. Test generation strategies

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.

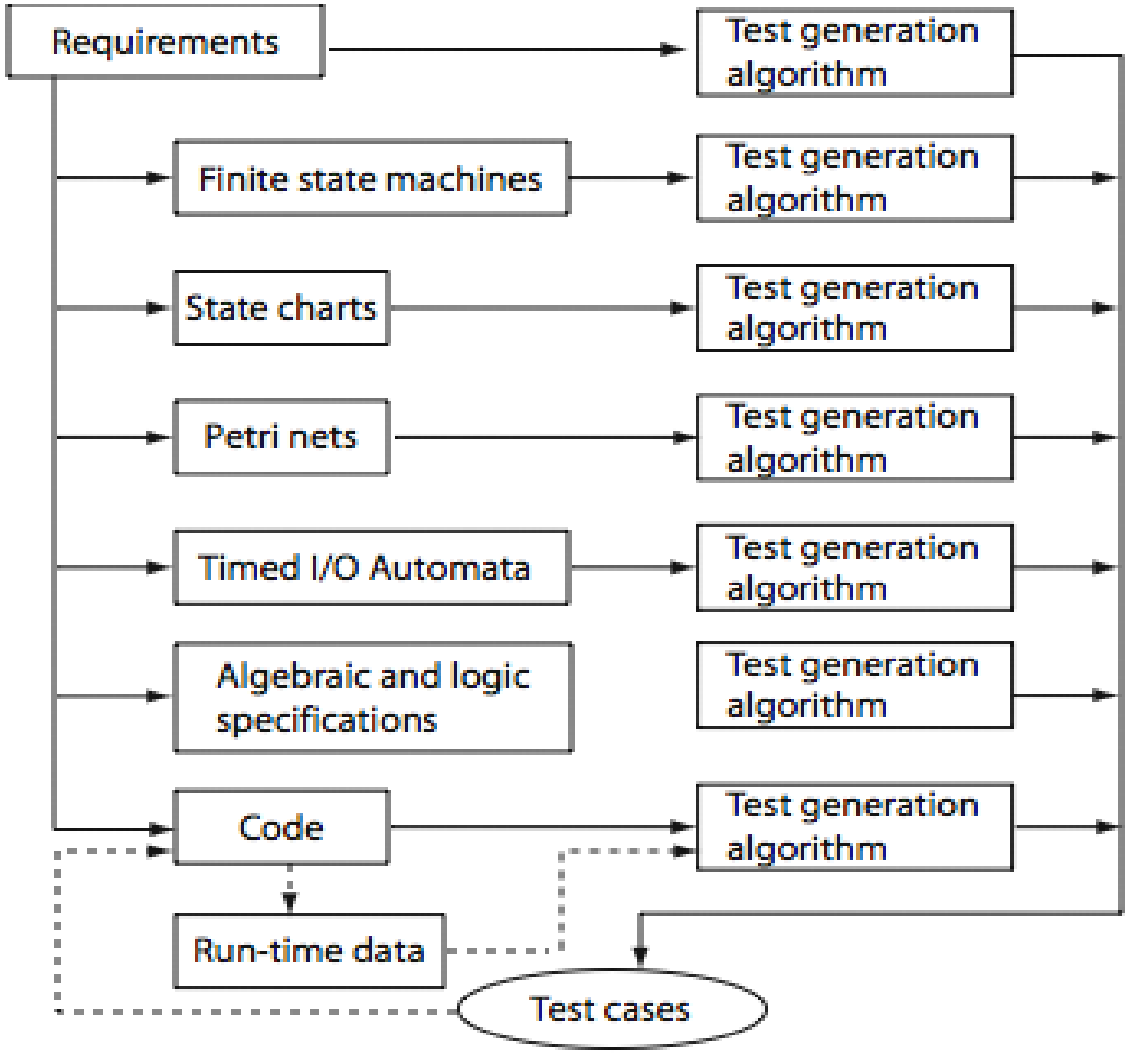
Test generation strategies

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.

Test generation strategies (Summary)



1.13 Types of software testing

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

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

C2: Lifecycle phase

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

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	

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

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