Software Testing: Basic Definitions

Lecture 1b

Agenda

- What is testing?
- Basic definitions
- Test cases
- Fault taxonomies

What is Software Testing?

What is Testing?

- Testing is the **process** of executing a program with the **intent** of **finding errors**.
- Testing is a set of **planned** activities that are conducted **systematically** to **uncover errors** that were made inadvertently as a software was **designed and constructed**.

Basic Definitions

Basic Definitions

- The International Software Testing Qualification Board (ISTQB) is a software testing certification board that operates internationally.
- The terminology here is compatible with the ISTQB definitions, and they, in turn, are compatible with the standards developed by IEEE.

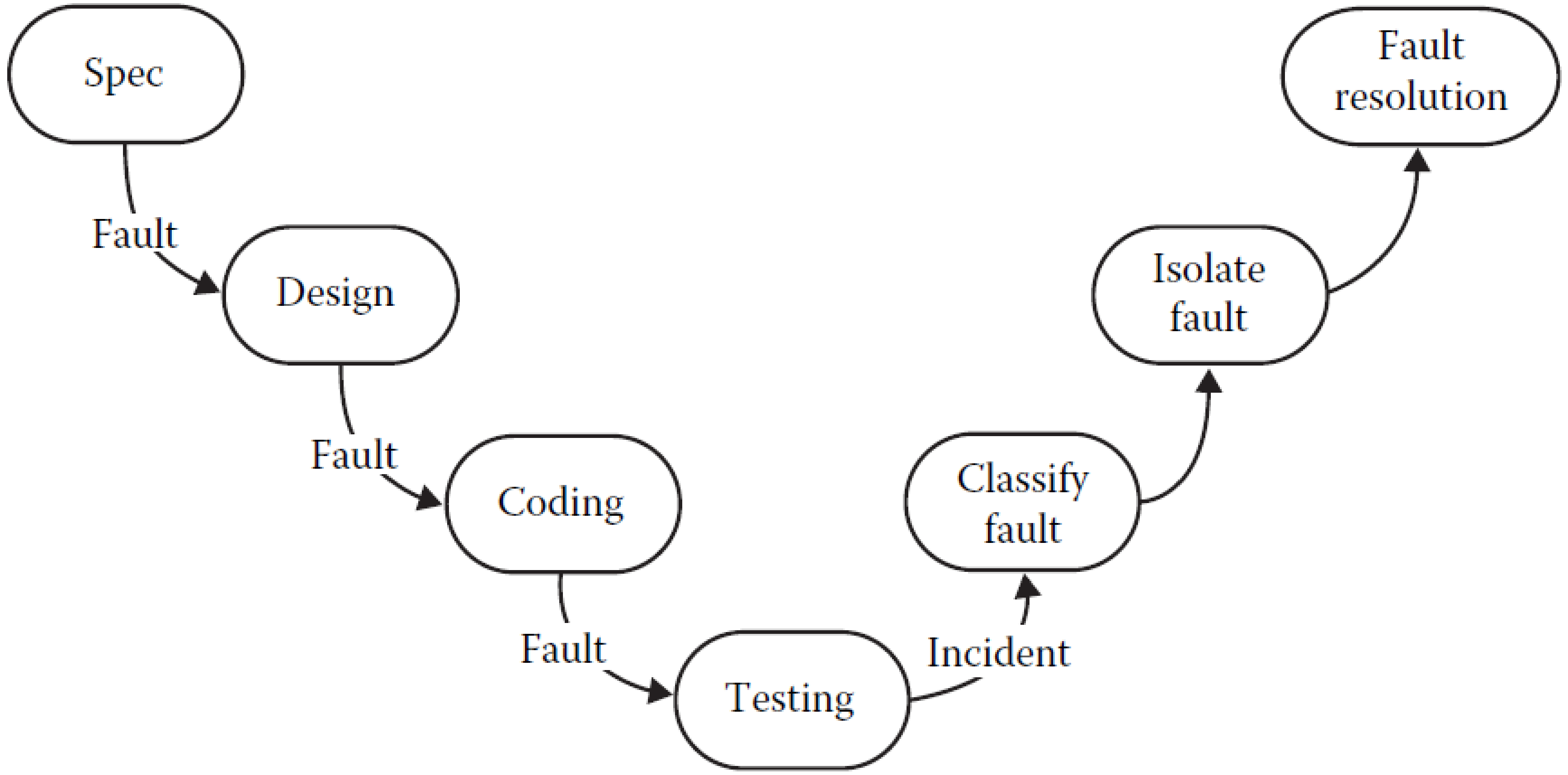
Basic Definitions

- **Error**: People make errors. A good synonym is *mistake*. When people make mistakes while coding, we call these mistakes *bugs*. Errors tend to propagate; a requirements error may be magnified during design and amplified still more during coding.
- **Fault:** A fault is the result of an error. It is more precise to say that a fault is the representation of an error. This representation can be a UML diagram or a source code. *Defect* is a good synonym for fault, as is *bug*.
    - **Faults of commission** : We enter something into a representation that is incorrect.
    - **Faults of omission**: We fail to enter correct information that should have been present in the representation, but it is missing.

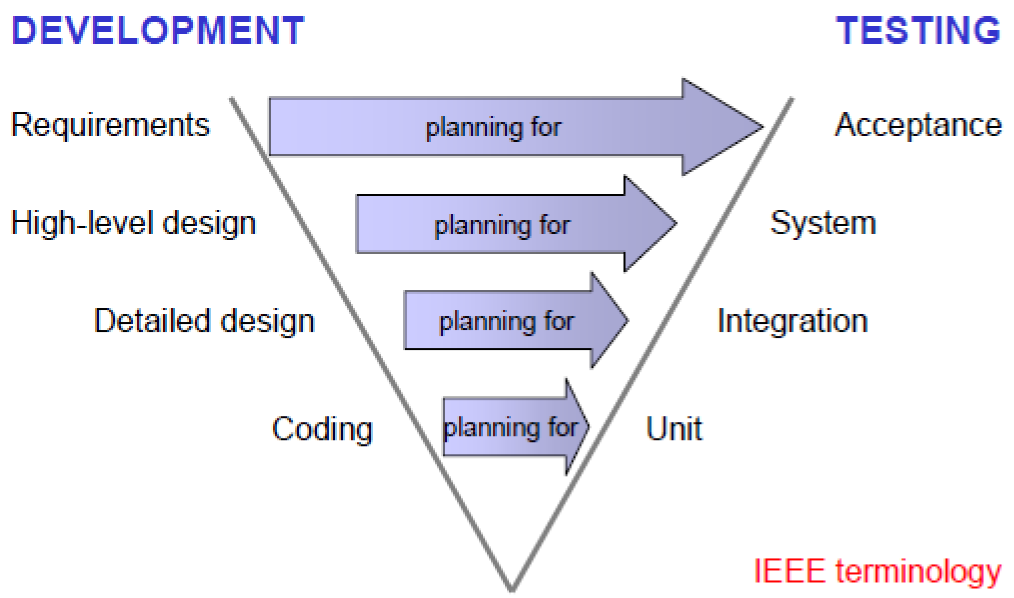
Basic Definitions

- **Failure:** A failure occurs when the code corresponding to a fault executes.
- **Incident:** An incident is the symptom associated with a failure that alerts the user to the occurrence of a failure.
- **Test:** A test is the act of exercising software with test cases. A test has two distinct goals: to find failures or to demonstrate correct execution.
- **Test case:** A test case has an identity and is associated with a program behaviour. It also has a set of inputs and expected outputs.

A Testing Life Cycle



The Life Cycle Model for Testing



- **High-Level Design**: technology platform, physical deployment, selection of major structural components, how the system communicates, concurrency issues, etc.
- **Detailed Design**: components are refined into classes, interfaces are realized, relationships between classes are specified, design patterns are identified and applied, design of components and their interfaces, etc.

Levels of Testing

- **Unit testing** is a level of software testing where individual units/ components of a software are tested.
- **Integration testing** is a level of software testing where individual units are combined and tested as a group.
- **System testing** is a level of software testing where a complete and integrated software is tested to verify that it meets specified requirements.
- **Acceptance testing** is a level of software testing where a system is tested for acceptability.

Levels of Testing

- Analogy
  - During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately, and **unit tested separately**. When two or more units are ready, they are **assembled**, and **Integration Testing** is performed. When the complete pen is integrated, **System Testing** is performed. Once System Testing is complete, **Acceptance Testing** is performed to confirm that the ballpoint pen is **ready** to be made available to the **end-users**.
  - Courtesy: http://softwaretestingfundamentals.com/acceptance-testing/

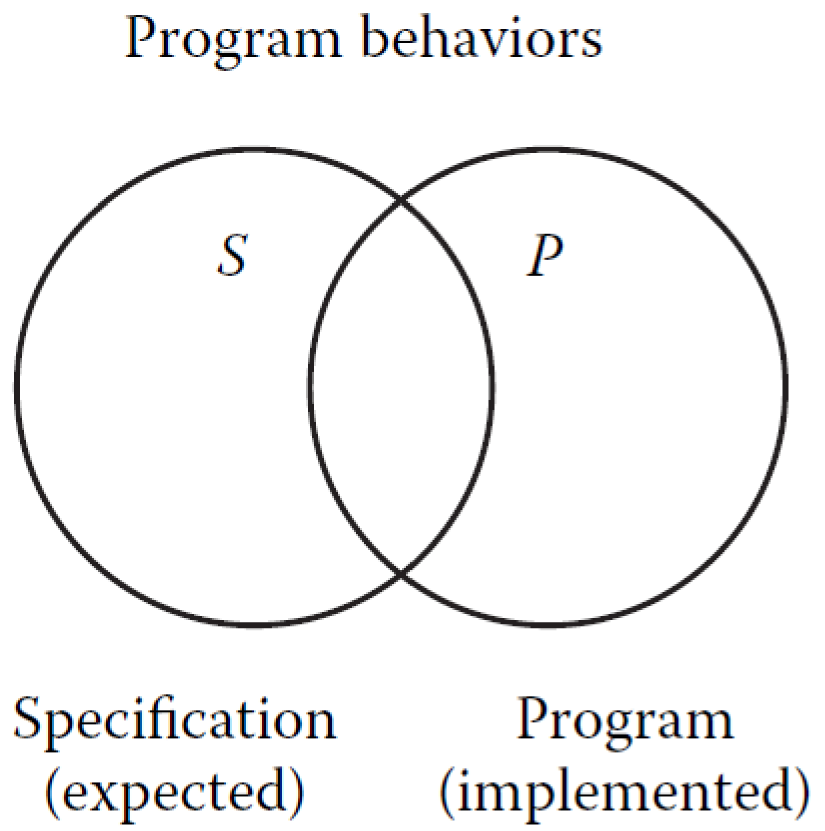
Test Cases

Test Cases

- The essence of software testing is to determine a set of test cases for the item to be tested.
- A test case is (or should be) a recognized work product.
- A complete test case will contain
  - a test case identifier,
  - a brief statement of purpose (e.g., a business rule),
  - a description of preconditions,
  - the actual test case inputs,
  - the expected outputs,
  - a description of expected postconditions,
  - and an execution history

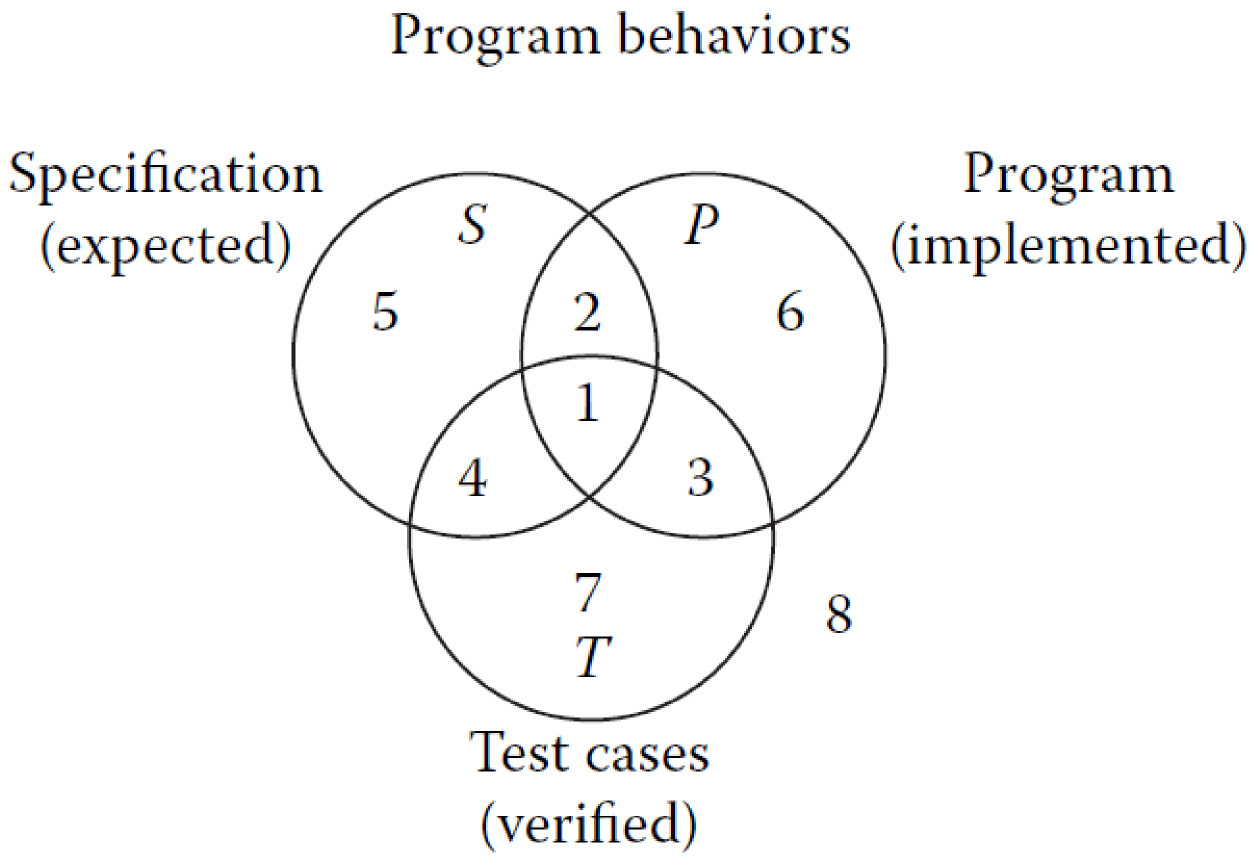
Test Cases Insights from a Venn Diagram

- Given a program and its specification, consider the set *S* of specified behaviours and the set *P* of programmed behaviours.
  - Fault of omission: ***S - P***
  - Fault of commission: ***P - S***
  - Correct portion: ***P ∩ S***



Test Cases Insights from a Venn Diagram

- Now, consider the relationships among sets *S*, *P*, and *T* of tested behaviours.
- Maximise $P \cap S \cap T$



Fault Taxonomies

Fault Taxonomies

- Partially based on the IEEE Standard Classification for Software Anomalies (IEEE, 1993).

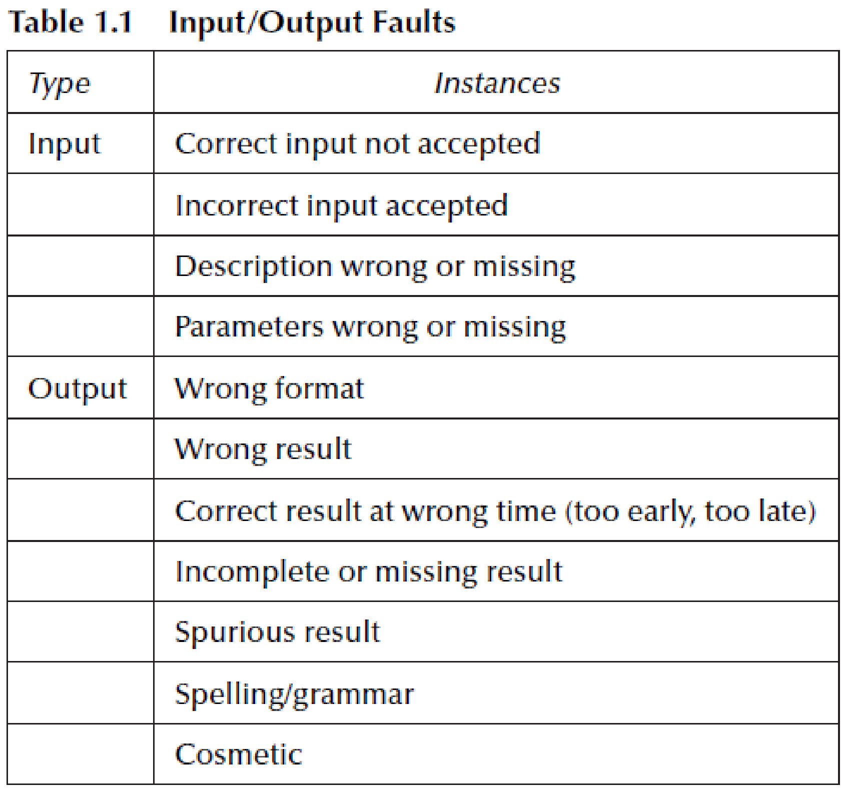


Table 1.1 Input/Output Faults

Type	*Instances*
Input	Correct input not accepted
	Incorrect input accepted
	Description wrong or missing
	Parameters wrong or missing
Output	Wrong format
	Wrong result
	Correct result at wrong time (too early, too late)
	Incomplete or missing result
	Spurious result
	Spelling/grammar
	Cosmetic

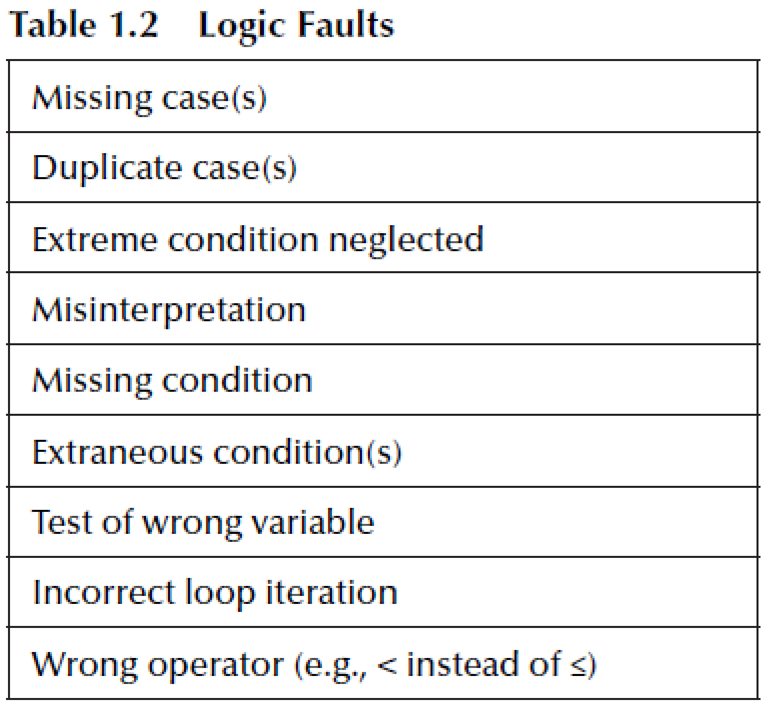


Table 1.2 Logic Faults

Missing case(s)
Duplicate case(s)
Extreme condition neglected
Misinterpretation
Missing condition
Extraneous condition(s)
Test of wrong variable
Incorrect loop iteration
Wrong operator (e.g., $<$ instead of $\leq$)

Fault Taxonomies

Table 1.3 Computation Faults

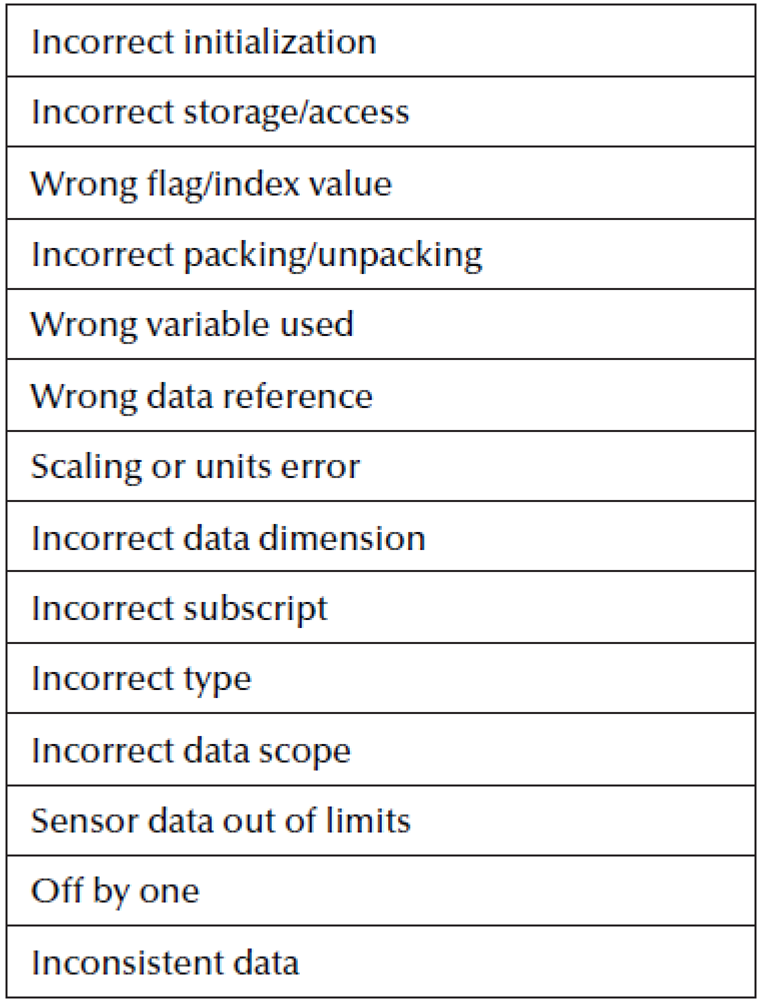
Incorrect algorithm
Missing computation
Incorrect operand
Incorrect operation
Parenthesis error
Insufficient precision (round-off, truncation)
Wrong built-in function

Table 1.4 Interface Faults

Incorrect interrupt handling
I/O timing
Call to wrong procedure
Call to nonexistent procedure
Parameter mismatch (type, number)
Incompatible types
Superfluous inclusion

Fault Taxonomies

Table 1.5 Data Faults



Incorrect initialization
Incorrect storage/access
Wrong flag/index value
Incorrect packing/unpacking
Wrong variable used
Wrong data reference
Scaling or units error
Incorrect data dimension
Incorrect subscript
Incorrect type
Incorrect data scope
Sensor data out of limits
Off by one
Inconsistent data

Next On

Take Away Points

- What is testing?
  - What is it intended for?
  - Why does it need to be systematic?
- Basic definitions
  - Error, fault, defect, bug, failure, incident.
  - How is the life cycle model for testing?
- Test cases
  - What does a test case contain?
  - Does it test specified or implemented behaviour? Partially or completely?
- Fault taxonomies
  - Input/Output, Logic, Computation, Interface and Data Faults.

Further Reading

- Paul Jorgensen, “Software Testing: A Craftsman’s Approach”
    - Chapter 1: A Perspective on Testing
    - Chapter 2: Examples

Next Lecture

- Identification of test cases: functional and structural approaches.