Test Cases Identification

Lecture 2a

Agenda

- Characteristics of a good test case
- Approaches to test cases identification
- Functional testing
- Structural testing
- Functional approach versus structural approach

Characteristics of a Good Test Case

The Fundamental Problem of Testing Software

- **Problem:**
  - We cannot test for everything.
  - No system can be completely tested.
  - The need to have a clever testing methodology.
- **Solutions:**
  - Test prioritisation based on risk analysis
    - What you test is more important than how much you test.
    - Tests need to be prioritised (likelihood and impact) so that the most important bugs are found first.
  - Tests must be carefully designed

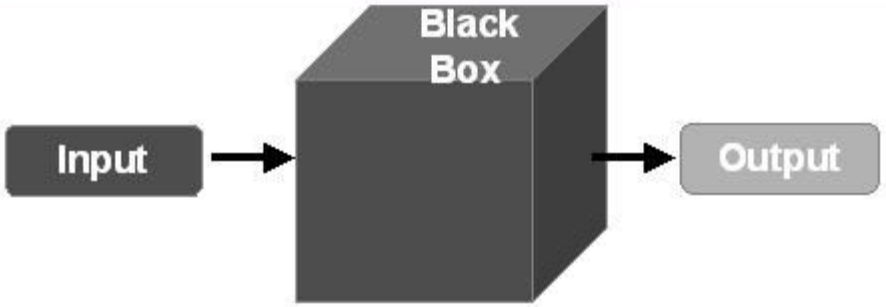
Characteristics of a Good Test Case

- A **high** probability of **finding** an **error**.
  - Develop a mental picture of how the software might fail.
- **Not redundant**.
  - Time and resources are limited. Then, every test should have a different purpose (even if it is subtly different).
- Should be "**best of breed**".
  - In a group of tests that have a similar intent, time and resource limitations may dictate the execution of only those tests that has the highest likelihood of uncovering a whole class of errors.
- Should be **neither** too **simple** nor too **complex**.
  - Each test should be executed separately.

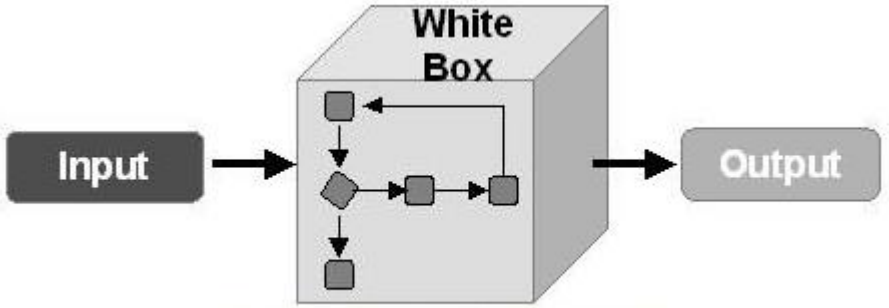
Approaches to Test Cases Identification

Approaches to Test Cases Identification

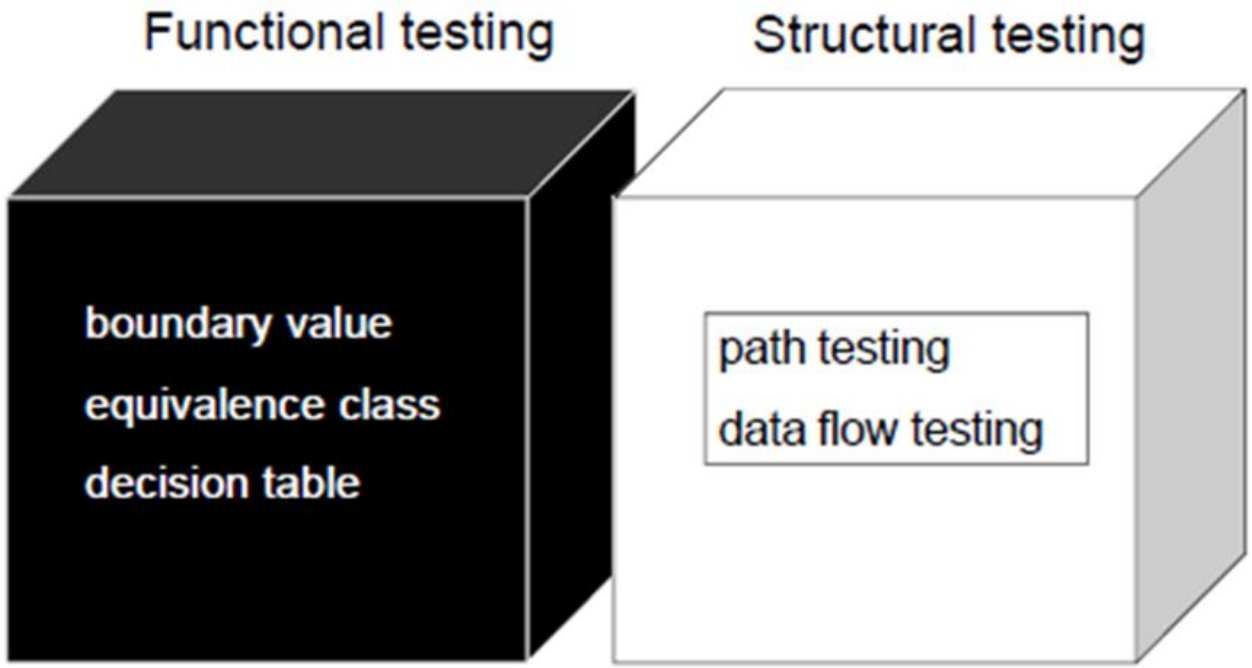
- Two fundamental approaches are used to identify test cases.
  - Knowing the specified **function** that a product has been designed to perform.



  - Knowing the **internal** workings of a product.



Approaches to Test Cases Identification



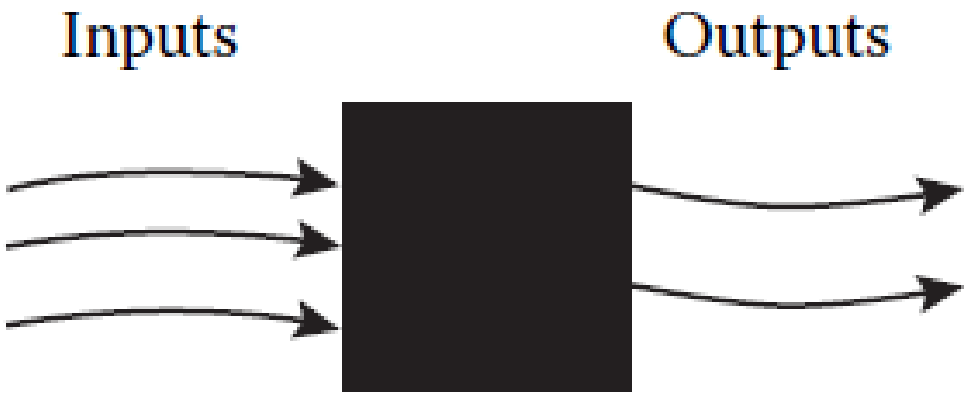
Approaches to Test Cases Identification

- **Functional testing**, also called specification-based testing or **black-box testing**, alludes to tests that are conducted at the software interface and focuses on the functional requirements of the software.
- **Structural testing**, sometimes called code-based, **white-box testing** or glass-box testing, is based on inner workings of an application and revolves around internal structure testing.

Functional Testing

Functional Testing

- Any **program** can be considered as a function that **maps values** from its input **domain to** values in its output **range**.
- Also known as black box testing because the implementation of the **software is not known**, and the function of the black box is understood completely in terms of its inputs and outputs.



Functional Testing

- In this approach, the only information used is the specification of the software.
  - Most people successfully operate automobiles with only black box knowledge.
- Black-box testing attempts to **find errors** in the following **categories**:
  1. Incorrect or missing functions.
  2. Interface errors.
  3. Errors in data structures or external database access.
  4. Behaviour or performance errors.
  5. Initialization and termination errors.

Functional Testing

- Functional tests cases are designed to **answer** the following questions:
  - How is functional validity tested?
  - How are system **behaviour** and **performance** tested?
  - What **classes of input** will make good test cases?
  - Is the system particularly **sensitive** to certain input values?
  - How are the boundaries of a data class isolated?
  - What **data rates** and **data volume** can the system tolerate?
  - What effect will specific combinations of data have on system operation?

Functional Testing

- **Advantages**:
  - Independence of how the software is implemented, so if the implementation changes, the test cases are still useful;
  - test case development can occur in parallel with the implementation, thereby reducing the overall project development interval.
- **Disadvantages**:
  - Functional testing often suffers from significant redundancies among test cases, compounded by the possibility of gaps of untested software.

Functional Testing

- Techniques:
  - **Boundary value:** leads to a selection of test cases that exercise bounding values.
  - **Equivalence class**: divides the input domain of a program into classes of data from which test cases can be derived.
  - **Decision table:** used when there are many possible combinations of conditions to test.

Structural Testing

Structural Testing

- It is sometimes called **white box** (or even clear box) testing or **code-based** testing.
- Based on the source code / pseudocode of the program or the system, and **NOT** on its specification.
- Primarily used for testing **imperative**-style programs/designs.
- Can be applied at different levels of granularity.

Structural Testing

- Using white-box testing methods, you can **derive test cases** that
  1. Guarantee that all **independent paths** within a module have been exercised at least once.
  2. Exercise all **logical decisions** on their true and false sides.
  3. Execute all **loops** at their boundaries and within their operational bounds.
  4. Exercise **internal** data **structures** to ensure their validity.

Structural Testing

- Because structural testing is based on the program, it is **hard** to imagine it can identify **behaviours** that are **not programmed**.
- It is easy to imagine, however, that a set of structural test cases is relatively small with respect to the full set of specified behaviours.

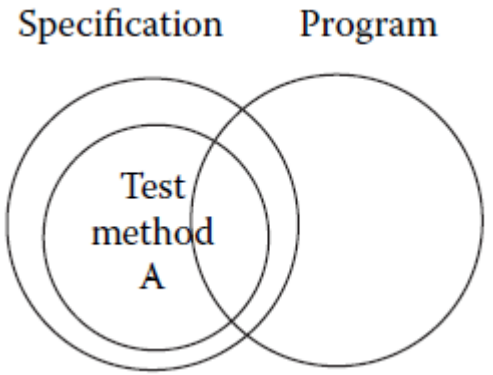
Structural Testing

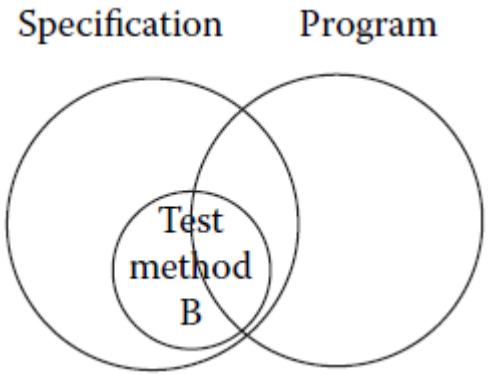
- Techniques:
  - **Path testing:** derives a program graph from a given program. A program graph is a directed graph in which nodes are statement fragments, and edges represent flow of control.
  - **Data flow testing:** focus on the points at which variables receive values and the points at which these values are used (or referenced).

Functional Approach versus Structural Approach

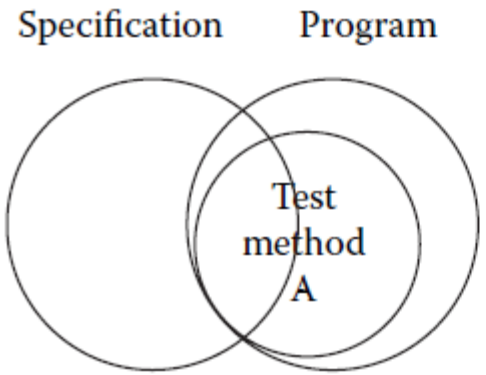
Functional Approach versus Structural Approach

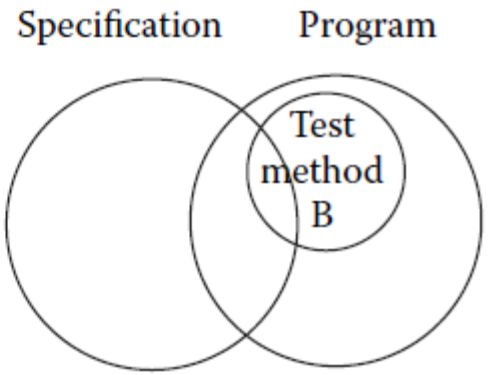
- Neither approach by itself is sufficient.
  - With functional techniques:





  - With structural techniques:





Functional Approach versus Structural Approach

- **Consider program behaviours**: if all specified behaviours have not been implemented, structural test cases will never be able to recognize this.
- **Conversely**, if the program implements behaviours that have not been specified, this will never be revealed by functional test cases.

Functional Approach versus Structural Approach

- Then, both approaches are needed.
- A judicious combination of approaches will provide the confidence of specification-based testing and the measurement of code-based testing.

Next On

Take Away Points

- Characteristics of a good test case
  - What is the fundamental problem of testing?
  - What do we mean by a good test case?
- Approaches to test cases identification
  - What are their key differences?
- Functional testing
  - Kind of errors attempted by this approach
  - What are their derived test cases aimed at?
- Structural testing
  - What are their derived test cases aimed at?
- Functional approach versus structural approach
  - Which one is better?

Further Reading

- Paul Jorgensen, “Software Testing: A Craftsman’s Approach”
  - Chapter 1: A Perspective on Testing
- Pressman and Maxin, “Software Engineering: A Practitioner’s Approach”
  - Chapter 23: Testing Conventional Applications

Next Lecture

- Continuous integration