Lecture 12 - Testing II - Execution-Based Testing

Collin Roberts

October 24, 2023

Outline

Execution-Based Testing

- Testing is a crucial part of any software development life-cycle.
- However we must keep in mind that (as Dijkstra points out), testing can demonstrate the presence of faults in a software product, **not** their absence.

Execution-Based Testing

One reason: Test cases are only as good as the tester selecting them. Things can get missed.

What Should Be Tested?

Definition 1

Execution-Based Testing *is a process of inferring certain behavioural properties of a software product based, in part, on the results of running the software product in a known environment with selected inputs.*

Three Troubling Details About Definition 1

1. Testing is an inferential process. There is no algorithm for determining whether faults are present! A test run with correct results may simply fail to expose a fault.

Three Troubling Details About Definition 1

2. What do we mean by **known environment**?

   We can never fully know our environment. The text gives the example that an intermittent hardware fault in the computer's memory system could cause failures, even if the code is perfect.

Three Troubling Details About Definition 1

3. What do we mean by **selected inputs**?

   With a real-time system, no control over the inputs is possible, e.g.

   1. an avionics system in an aircraft, for which the inputs describing the current state of the aircraft's flight cannot be controlled (a partial solution to this problem is provided by a **simulator**), and
   2. a system for controlling trains.

Remarks

1. Despite these problems, Definition 1 is the best one available.

Utility

Definition 2

The **utility** *of a software product is the extent to which the software product meets the user's needs when operated under conditions permitted by its specification.*

Elements

1. Is the software product easy to use?
2. Does the software product perform useful functions?
3. Is the software product cost effective?

Remarks

1. If a software product fails a test of its utility, then testing should proceed no further!

Reliability

Definition 3

The **reliability** *of a software product measures the* **frequency** *and* **severity** *of its failures.*

Elements

1. **mean time between failures** Long times → more reliable.
2. **mean time to repair failures** Long times → less reliable.
    1. **Also important (often overlooked):** time required to fix the **effects** of the failure (e.g. correcting corrupted data). Long times → less reliable.

Elements

1. range of operating conditions (permissible by the specifications, or not)
    1. A robust product has a wide range of operating conditions, including some outside its specification.
2. possibility of unacceptable output given acceptable input
    1. A robust product produces acceptable output, given acceptable input.
3. acceptability of output given unacceptable input
    1. A robust product produces acceptable output (e.g. a helpful error message instead of a crash), even given unacceptable input.

Performance

1. It is crucial to verify that a software product meets its constraints with respect to:
    1. Space constraints which can be critical in miniature applications, e.g.
        1. missile guidance systems as in the text, or
        2. smart phone apps.
    2. Time constraints which can be critical in **real time** applications, e.g.
        1. measuring core temperature in a nuclear reactor as in the text, or
        2. controlling signals on a railroad network.

Correctness

Definition 4

A software product is **correct** *if it satisfies its output specification, without regard for the computing resources consumed, when operated under permissible (pre-)conditions.*

Remarks

1. This definition is **partial correctness**. It tacitly assumes that the program **terminates**.

Problems with Definition 4

1. Specifications can be wrong.
    1. Then a software product can be correct, but not be acceptable.
        1. Cute text example: a sort program whose specification omits the requirement that the sorted list be a permutation of the original list - clearly not acceptable!
2.
    1. A software product can be acceptable, but not be correct.
        1. Cute text example: a compiler, faster than its predecessor, but which prints a spurious error message (which is easily ignored) in one rare situation. This compiler is acceptable. However it is not correct since producing the spurious error message is not part of its specification.