Introduction to Software Testing

COS214

Andrew Broekman

University of Pretoria

2022

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Outcomes

After this lecture you should be able to:

- Understand various types of testing
- Apply unit testing
- Design test cases

Application Programming Interface

- External API - RESTful/ XML/ SOAP
- Internal API - Interfaces/Contracts/Class

Define and Implement Service

- Interface
- Service name $\rightarrow$ Class name
- Operation name $\rightarrow$ Function name
- Operation has preconditions/postconditions
- Exceptions that can be thrown

Failure of Testing

- Therac-25 bug - Delivered more than 100 time intended dose of radition to patients. Two died
- Ariane 5 rocket - 37 seconds after launch destroyed, costing US$370 million. Error in 64-bit floating point to 16-bit signed conversion error.
- 1,200 US Veterans wrongly informed they had fatal Lou Gehrig's neurological disease

Failure of Testing

- MIM-104 Patriot - System clock drift - resulting in failure to intercept an incoming missile, killing 28 Americans
- Sony BMG rootkit scandal - Installing rootkits on Windows machines

Characteristics

- Start from inside and work towards the outside
  - Function level
  - Class level
  - Between classes
  - Between modules/namespaces
  - Between systems
- Different techniques
- Testing $\neq$ Debugging (they do function together)

Creating & Destroying

- Dichotomy — "a division or contrast between two things that are or are represented as being opposed or entirely different." (Oxford Languages)
- **Building** software - Creating
- **Testing** software - Destroying

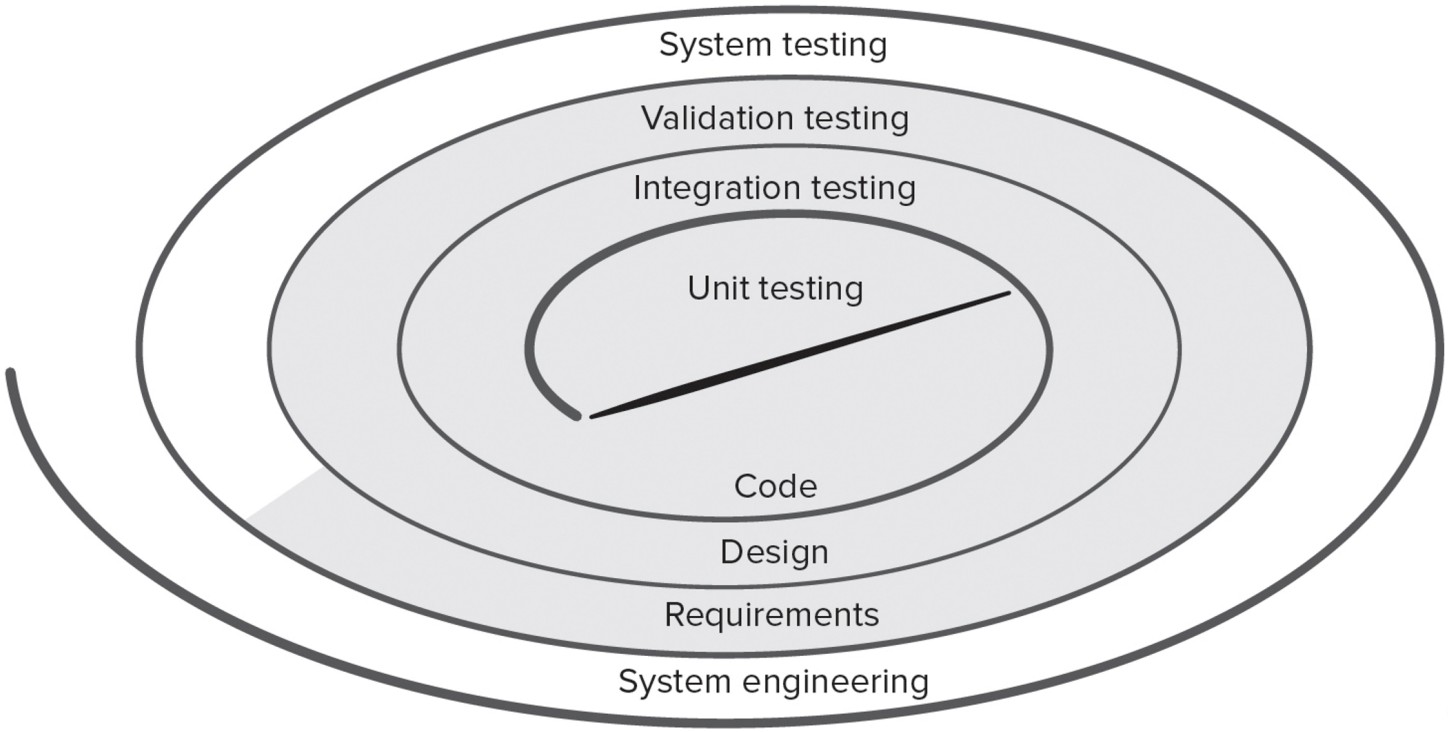
Why Test?

- Massive effort
- Often the most time & cost of any software system
- Ensuring that we build the correct software
- Part of good software quality (Software Quality Assurance)

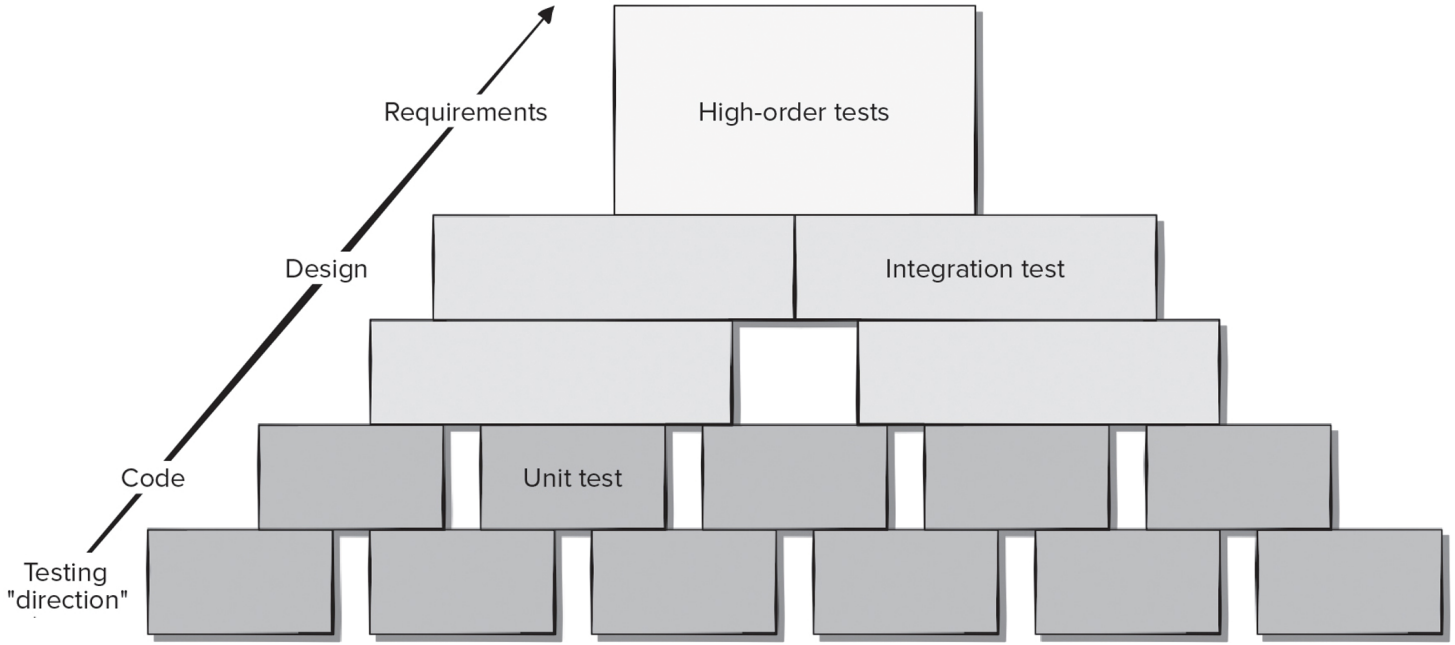
What Testing?

- Abstract Solution (Too an extent)
- Implemented/Realised solution
- Smallest to largest component

How to test?



Steps to test?



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Types of tests?

- *Unit testing* — Small units, e.g. classes, component, functions
- *Integration testing* — Does it all fit together and WORK?
- *Validation testing* — Requirements are checked against the software
- *Regression testing* — Retesting components that may be affected by changes
- *System testing (e2e)* — Tested as one unit

Testing Frameworks

- Automated testing framework
  - Microsoft Unit Testing Framework for C++
  - Google Test
  - Boost.Test
  - CTest

Effective Testing

- Exhaustive testing - Not feasible
- Cost/Tome of exhaustive testing not always worth it
- Test smart — Crucial modules, error-prone modules

Test Case Design

- Testing module interface
- Local data structures stored data maintains integrity
- Independent paths are tested (Branch coverage)
- Boundary conditions
- All error-handling paths

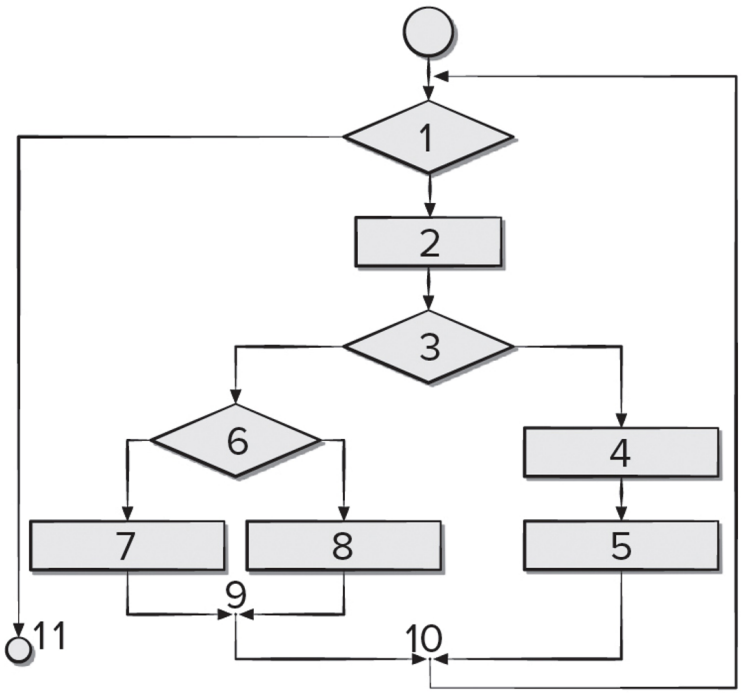
Traceability

- Each test case should be traceable to a preconditions, postconditions or exceptions
- Traceable back to our API design
- Test failures often due to missing traceability, inconsistent test data, incomplete coverage

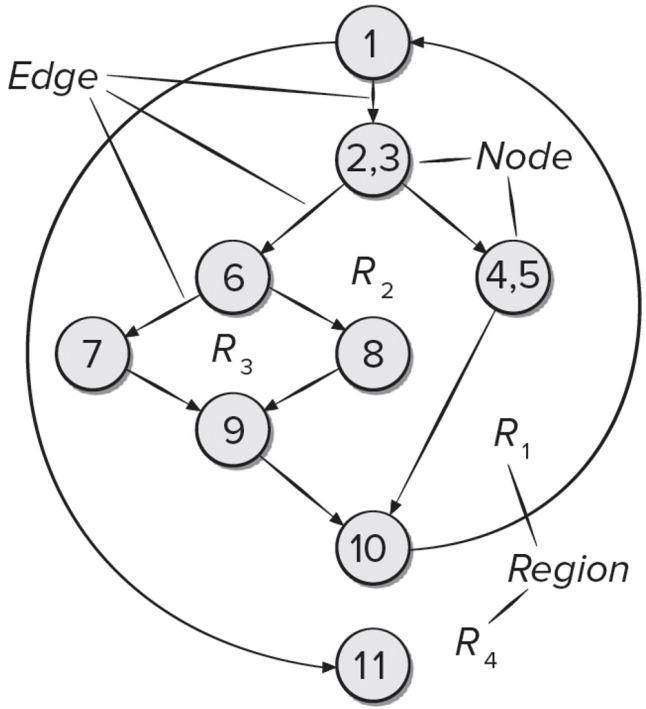
White-box Testing

- Guarantee that all independent paths within a module have been exercised at least once
- Exercise all logical decisions on their true and false sides
- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

Basis Path Testing



(a)



(b)

Basis Path Testing

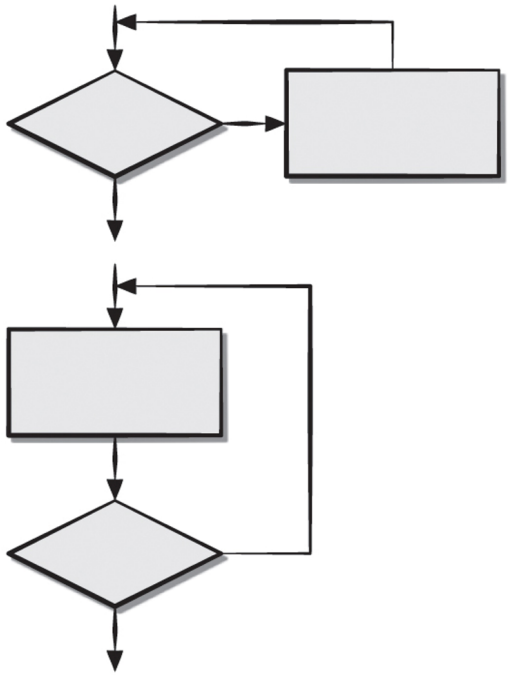
- *Independent path* — Any path through the program that introduces **at least one** new set of processing statements or a new condition
- Path 1: 1-11
- Path 2: 1-2-3-4-5-10-1-11
- Path 3: 1-2-3-6-8-9-10-1-11
- Path 4: 1-2-3-6-7-9-10-1-11

Basis Path Testing - Design Test Cases

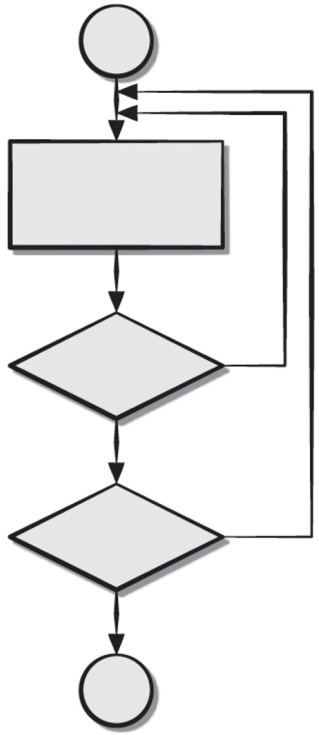
- Draw a corresponding flow graph
- Determine a basis set of linearly independent paths
- Prepare test cases that will force execution of each path in the basis set

Control Structure Testing

- *Condition testing* — Method that exercises the logical conditions
- *Data flow testing* — Selects test paths of a program according to variables
- *Loop testing* — Focuses exclusively on the validity of loop constructs



Simple loops



Nested loops

Simple Loop Testing

- Skip the loop entirely
- Only one pass through the loop
- Two passes through the loop
- m passes through the loop where $m < n$
- $n1, n, n + 1$ passes through the loop

Boundary Value Analysis

- Exercise bounding values.
- Range $[a, b]$ — Test with values a and b and just above and just below a and b
- n values — Test cases should exercise the min and max numbers as well as values just above and below min and max
- Apply above guidelines to both input and output
- Internal data structures have prescribed boundaries (array with max index of 100) — Design test case to exercise the data structure at its boundary

Overview

Attempts to find errors in following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Environment

- CMake
- GoogleTest - `https://github.com/google/googletest`
- Visual Studio Code
- Ubuntu

Install GoogleTest on Ubuntu

- `sudo apt-get install libgtest-dev`
- `sudo apt-get install cmake`
- `cd /usr/src/gtest`
- `sudo cmake CMakeLists.txt`
- `sudo make`
- `sudo cp ./lib/libgtest*.a /usr/lib`

Other OS's

- https://alexanderbussan.medium.com/getting-started-with-google-test-on-os-x-a07eee7ae6dc
- https://medium.com/swlh/google-test-installation-guide-for-c-in-windows-for-vi

Examples

- Sum - Boundary Value Analysis
- Prime - Conditional/Branch testing
- Factorial - Loop testing

Examples Compile

- `cmake CMakeLists.txt`
- `make`
- `./runTests`

Homework

Go and do research on the following acronyms in software testing? Which approach do you prefer? What are the advantages and disadvantages?

- TDD
- BDD
- ATDD

Questions

Any questions with regards to:

- Testing
- Code examples