SENG3320 – Notes

Week One

Bryce Tuppurainen

**Fault**: incorrect portions of code.

**Failure**: observable incorrect behaviour of a program

**Error**: cause of a fault. Whether it was a typo, conceptual, cognitive, etc

**Bug**: an informal term to describe a fault

**Test Case**: A series of inputs, and/or execution conditions, with a series of linked expected results developed for a particular objective. Such as an exercise to a particular program ‘path’ to ensure that it is running to completion and achieving the correct result. This can either be automated; or can literally be a user testing a functionality within reasonable use, such as using the different features of an email client.

**Testing**: Finding a bug. Often through executing and analysing test cases

**Debugging**: Finding the cause of the bug (therefore, the bug must be known beforehand)

Exhaustive testing

In exhaustive testing, you take the sample size {(), … , (n)} of all possible input permutations. Therefore, if you’ve got two 32-bit integers available that can be any value, then the exhaustive testing would require having to use 2^32 x 2^32 = 2^64 possible input permutations, i.e. 18,446,744,073,709,551,616 values. If you attempted one per second, every second, it would take > 9,000,000,000 years.

Black Box Testing – No provided source (or analysed source)

Falls into two major techniques. Which are equivalence partitioning and boundary-value analysis (a subset of EP)

Equivalence Partitioning (EP)

Divide the input domain into unique equivalence partitions and then ensure that you’ve got at least a minimum of one test case from each partition and that partitions cover every possible value in the input sample space. Equivalence partitions are efficient if they’re disjointed and provide sufficient coverage, i.e. no overlap between partitions, and the entire input domain is covered by the sum of all partitions

Boundary-Value Analysis (BVA)

A subset of equivalence partitioning, instead you select values near the boundaries of partitions of data instead, typically as well as on the boundary. So for a password requirement of a length of 8-16 characters, you’d divide this into the partitions and boundaries <8, 8, 8-16, 16, >16 then test at least one test case from each.

White Box Testing – Provided Source (which is then analysed)

In white box testing, we generate test cases based on the current design of a program. This often relates to modelling the code into a CFG (Control Flow Graph which is essentially just a model of the state machine).

Coverage Metrics – Memorise SBP

Ensure that every single line is executed to ensure that there are no missed or unneeded statements and that all the program paths can run the whole way through.

**Statement coverage**: **all statements** in the programs should be executed at least once

**Branch coverage**: **all branches** in the program should be executed at least once

**Path coverage**: **all execution paths** in the program should be executed at least once

CFG for various logic – Essentially graphical pseudocode

Diagram

Description automatically generated

*Figure 1.0 - Representation of a CFG for an if statement*

­­ Diagram

Description automatically generated

*Figure 1.1 – Representation of a CFG for a loop (while, do-while, for, goto, etc)*

Diagram

Description automatically generated

*Figure 1.2 – Representation of a CFG for a switch case*

The rules are essentially that **every node has one entrypoint and one exit point** which can have **any number of edges coming from or to it respectively with arrows**, typically working top-to-bottom. The **nodes state the actions which occur** within these lines and the **edges store the arguments that determine which edge is followed** at the exit of a node.