B07 Nov 29 Lec 1 Notes

Testing Levels

- Unit testing
  - Test units (methods) individually.
  - Easiest form of testing
- Module testing
  - A module is a collection of related units that are assembled in a file, package, or class.
  - Test modules in isolation including how the components interact with each other.
- Integration testing
  - Tests how modules interact with each other.
- System Testing
  - Test the overall functionality of the system.
- Acceptance Testing
  - Test whether the software is acceptable to the user.

Black / White - Box Testing

- Black - Box Testing
  - Test are derived from external descriptions of the software
- White - Box Testing
  - Test are derived from the soure code internals of the software
  - More expensive to apply.

Software Testing is Hard

- Exhausive testing is infeasible
  - Exhausively testing a method with two integer parameters would require $\sim 10^{19}$ tests.
- Random / statistical testing is not effective.

Fault / Error / Failure

- Software Fault : A static defect in the software.
  - e.g. a semantic defect, but the code can be compiled
- Software Error : An incorrect internal state that is the manifestation of some fault
- Software Failure: External, incorrect behaviour with respect to the requirements or another description of the expected behaviour.

The RIPR Model

- Four conditions are needed for a failure to be observed.

(i) Reachability: a test must reach the location in the program that contains the fault.

(ii) Infection: After the faulty location is executed, the state of the program must be incorrect

(iii) Propagation: The infected state must propagate through the rest of the execution and cause some output or final state of the program to be incorrect.

(iv) Revealability: The tester must observe part of the incorrect portion of the final program state.

Criteria-based Test Design

- Coverage Criterion: A rule or collection of rules that impose test requirements on a test set.
  - e.g. for each statement in the code, there should be at least one test case that covers it.
- Coverage criteria give us structured, pratical ways to search the input space. Satisfying a coverage criterion gives a tester some amount of confidence in two crucial goals.
  (i) We have looked in many corners of the input space, and
  (ii) Our tests have a fairly low amount of overlap.
- Criteria subsumption
  - C_1 subsumes C_2 iff every test set that satisfies C_1 satisfies C_2.

Criteria-based Test Design - Graph coverage

- The software is modeled as a graph where nodes and edges could represent:
  - Methods and calls
  - Statements and branches, etc.
- Coverage criteria are based on the graph. For example:
  - Cover every node
  - Cover every edge
  - Cover every path, etc.

Criteria-based Test Design - Logic coverage

- Involves the boolean expressions of the code.
- Coverage criteria include
  - Predicate coverage
    - The test suite should make each predicate evaluate to T/F.
  - Clause coverage
    - The test suite should make each clause evaluate to T/F.
- Clause coverage has a weakness - the values do not always make a difference.
- Active clause coverage.
  - A clause C_i in predicate p, called the major clause, determines p iff the values of the remaining minor clauses c_j are s.t. changing c_i changes the value of p.
  - Two requirements for each C_i: C_i evaluate to true and c_i evaluates to false
  - This is a form of MCDC, which is required by the FAA for safety critical software.
- Inactive clause coverage
  - Ensures that "major" clauses do not affect the predicate.

Test Oracles

- A test oracle is an encoding of the expected results of a given test.
- There must be a balance between checking too much (unnecessary cost) and checking too little. (perhaps not revealing failures)
- How do we determine what the correct results are?
  - Specification-Based direct verification of outputs
    - e.g. "a sort program should produce a permutation of its input in increasing order"
  - Redundant computations
    - Refer to another trustworthy implementation of the program.
    - Usually used for regression testing
  - Consistency checks
    - Check whether certain properties hold (e.g. a value representing probability should neither be negative nor larger than one.