Chapter 10
Software Testing

Text Book

- Software Engineering: Principles and Practice
- By Hans van Vliet
- 3rd Edition
- Wiley India

Test adequacy criteria

- Specifies requirements for testing
- Can be used as *stopping rule*: stop testing if 100% of the statements have been tested
- Can be used as *measurement*: a test set that covers 80% of the test cases is better than one which covers 70%
- Can be used as *test case generator*: look for a test which exercises some statements not covered by the tests so far
- A given test adequacy criterion and the associated test technique are opposite sides of the same coin

Data Flow Testing

Adequacy Criteria

- **All-Uses coverage**
  - – tests which traverse a definition-clear path between each definition of a variable to each (P- or C-) use of that definition and each successor of that use.
- **All-DU-Paths coverage**
  - – slightly stronger criterion requires that each definition-clear path is either cycle-free or a simple cycle

Data Flow Testing [contd]

- **All-defs coverage** simply requires the test set to be such that each definition is used at least once.
- **All-C-uses/Some-P-uses coverage** requires definition-clear paths from each definition to each computational use. If a definition is used only in predicates, at least one definition-clear path to a predicate use must be exercised.

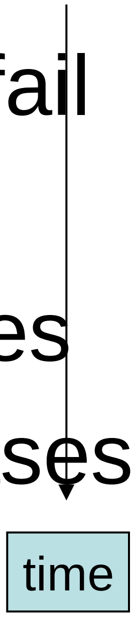
Data Flow Testing [contd]

- **All-P-Uses/Some-C-uses coverage** requires definition-clear paths from each definition to each predicate use. If a definition is used only in computations, at least one definition-clear path to a computational use must be exercised.
- **All-P-Uses coverage** requires definition-clear paths from each definition to each predicate use.

Testing models

- *Demonstration*: make sure the software satisfies the specs
- *Destruction*: try to make the software fail

- *Evaluation*: detect faults in early phases
- *Prevention*: prevent faults in early phases



When to stop?

- Stop if the test budget has run out

- Stop if all the test cases have been executed without any failures occurring

Testing and the life cycle

- requirements engineering
  - criteria: completeness, consistency, feasibility, and testability.
  - typical errors: missing, wrong, and extra information
  - determine testing strategy
  - generate functional test cases
  - test specification, through reviews and the like
- design
  - functional and structural tests can be devised on the basis of the decomposition
  - the design itself can be tested (against the requirements)
  - formal verification techniques
  - the architecture can be evaluated

Testing and the life cycle (cnt'd)

- implementation
  - check consistency implementation and previous documents
  - code-inspection and code-walkthrough
  - all kinds of functional and structural test techniques
  - extensive tool support
  - formal verification techniques
- maintenance
  - regression testing: either retest all, or a more selective retest

Test-Driven Development (TDD)

- *First* write the tests, *then* do the design/implementation
- Part of agile approaches like XP
- Supported by tools, eg. JUnit
- Is more than a mere test technique; it subsumes part of the design work

TDD [contd]

The way of working in each iteration of test-driven development consists of the following steps:

1. Add a test

2. Run all tests, and observe that the one added will fail

3. Make a small change to the system to make the test work

4. Run all tests again, and observe that they run properly

5. Refactor the system to remove duplicate code and improve its design.