Software Testing

Stuart Anderson

Room 1610, JCMB, KB

0131 650 5191, soa@inf.ed.ac.uk

Reading/Activity

- Please read pages 69—86 Of the SWEBOK for an overview of Software Testing.
- Please read the article: What is Software Testing? And Why is it so Hard?, James A. Whittaker, IEEE Software Jan/Feb 2000, 70—79.
- Acknowledgement: these slides were originated by Karine Arnout at ETH – I am responsible for any bugs introduced.

What is software testing?

- [Software testing] is the design and implementation of a special kind of software system: one that exercises another software system with the intent of finding bugs.

Robert V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools (1999)

What is software testing?

- Testing software typically involves:
  - Executing software with inputs representative of actual operation conditions
  - Comparing produced / expected outputs
  - Comparing resulting / expected states
  - Measuring execution characteristics (memory used, time consumed, etc.)

Terminology

- Fault:
  - An imperfection that may lead to failure. e.g. missing / incorrect code that may result in a failure
- Error:
  - Where the system state is incorrect but it may not have been observed
- Failure:
  - Some failure to deliver the expected service that is observable to the user
- Bug:
  - Another name for a fault in code

A few more definitions

- Test case: Set of inputs, execution conditions, and expected results developed for a particular objective.
- Test suite: Collection of test cases, typically related by a testing goal or an implementation dependency.
- Test driver: Class or utility program that applies test cases to an IUT.
- Test harness: System of test drivers and other tools that supports test execution.

A few more definitions (cont'd)

- Test strategy: Algorithm or heuristic to create test cases from a representation, implementation, or a test model.
- Oracle: Means to check the output from a program is correct for the given input.
- Stub: Partial temporary implementation of a component (usually required for a component to operate).

Effectiveness vs. Efficiency

- Test effectiveness:
  - Relative ability of testing strategy to find bugs in the software.
- Test efficiency:
  - Relative cost of finding a bug in the software under test.

What is a successful test?

- Pass:
  - Status of a completed test case whose actual results are the same as the expected results
- No pass:
  - Status of a completed test case whose actual results differ from the expected ones
  - "Successful" test (I.e. we want this to happen)

What software testing is NOT…

- Model verification (e.g. by simulation)
- Tool-based static code analysis
- Human documentation/code scrutiny
- Debugging:
  - Testing is NOT debugging, and debugging is NOT testing.

Summary

- The scope of testing:
  - The different levels of the system that testing addresses
- Test techniques:
  - Some of the approaches to building and applying tests
- Test management
  - How we manage the testing process to maximise the effectiveness and efficiency of the process for a given product.

Testing scope

- "Testing in the small" (unit test):
  - Exercising the smallest executable units of the system.
- "Testing the build" (integration test):
  - Finding problems in the interaction between components.
- "Testing in the large" (system test):
  - Putting the entire system to the test.

Testing “in the small”

- Unit Testing:
  - Exercising the smallest individually executable code units.
  - Objectives:
    - Find faults in the units.
    - Assure correct functional behavior of units.
  - Usually performed by programmers.

Testing the build

- Integration Testing:
  - Exercising two or more units or components.
  - Objectives:
    - Detect interface errors.
    - Assure the functionality of combined units.
  - Performed by programmers or testing group.
  - Issues:
    - Strategy for combining units?
    - Compatibility with third-party components?
    - Correctness of third-party components?

Testing “in the large”: System

- System Testing:
  - Exercising the functionality, performance, reliability, and security of the entire system.
  - Objectives:
    - Find errors in the overall system behavior.
    - Establish confidence in system functionality.
    - Validate non-functional system requirements.
  - Usually performed by a separate test group.

Testing “in the large”: accept

- Acceptance Testing:
  - Operating the system in the user environment with standard user input scenarios.
  - Objectives:
    - Evaluate whether the system meets the customer criteria.
    - Determine whether the customer will accept the system.
  - Usually performed by the end user.

Testing “in the large”: operation

- Regression Testing:
  - Testing modified versions of a previously validated system.
  - Objective: Assuring that changes to the system have not introduced new errors.
  - Performed by the system itself or by a regression test group.
  - Capture / Replay (CR) tools

Testing categorization

- Fault-directed testing:
  - Unit testing
  - Integration testing
- Conformance-directed testing:
  - System testing
  - Acceptance testing

Test generation methods

- Black-box testing:
  - No knowledge of the software structure
  - Also called specification-based or functional testing.
- White-box testing:
  - Knowledge of the software structure and implementation.
- Fault-based testing:
  - Objective is to find faults in the software.
  - e.g. Unit testing
- Model-based testing:
  - Use of a data or behavioral model of the software.
  - e.g. Finite state machine
- Random testing

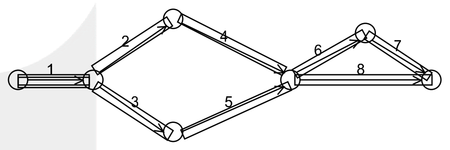
White-box Testing

- White-box methods can be used for:
  - Test generation
  - Test adequacy analysis
- Usually used as adequacy criteria (after generation by a black-box method).

White-box Testing (cont'd)

- Methods based on internal code structure:
  - Statement coverage
  - Branch coverage
  - Path coverage
  - Data-flow coverage

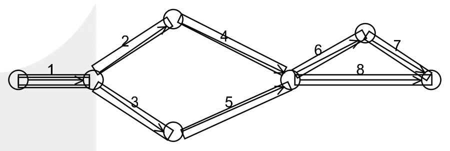
Branch Coverage



2 test cases: 12467; 1358

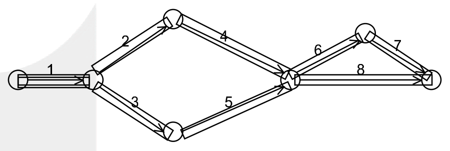
○ Statement
→ Branch

Path Coverage



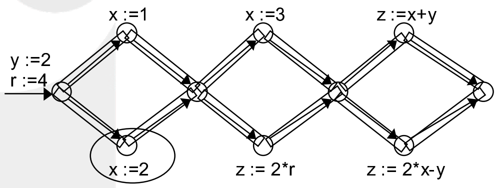
○ Statement
→ Branch

Path Coverage



4 test cases: 12467; 1358; 1248; 13567

Data-flow Coverage (All-uses)



Red path covers the definitions y :=2; r :=4; x :=1

Blue path covers the definitions y :=2; r := 4; x :=3

White-box Testing (cont'd)

- Issues:
  - Is code coverage effective at detecting faults?
  - How much coverage is enough?
  - Is one coverage criterion better than another?
  - Is coverage testing more effective than random test case selection?

Experimental studies

- Black-box generation followed by white-box coverage-based tests.
- Results:
  - High coverage alone does not guarantee fault detection.
  - Fault detection increases significantly as coverage goes above 95%.
  - No significant difference between Branch and Data-flow coverage.
  - Both Branch and Data-flow coverage are significantly more effective than random test cases.

Hutchins et al. "Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria". ICST, May 1994.

Test Management

- Management concerns
  - Attitude to testing.
  - Effective documentation and control of the whole test process
  - Documentation of tests and control of the test codebase
  - Independence of test activities.
  - Costing and estimation of test activities
  - Termination: deciding when to stop.
  - Managing effective reuse

Test Management (ctd)

- Test Activities
  - Test Planning
  - Test case generation – can involve massive amounts of data for some systems.
  - Test environment development
  - Execution of tests
  - Evaluating test results
  - Problem reporting
  - Defect tracking

Summary

- Testing is a critical part of the development of any system.
- Testing can be carried out at a number of levels and is planned as an integral part of the development process.
- There is a wide range of approaches to test case generation and evaluation of the adequacy of a test suite.
- Test needs to be managed effectively if it is to be efficient.