

# Software Testing Course Review

(version 1.4)

Conrad Hughes

University of Edinburgh

March 20, 2009

# Outline

- 1 Lecture review
- 2 Big themes
- 3 Exam

# Introduction

- What is testing?
- What can we test?
- What can testing permit us to say?
- Issues:
  - Modelling environment
  - Verifying results (oracle)
- What makes one test better than another?
- Measurement
- Lifecycle

# Types of testing

- What do we look at?
  - Specification (“black box”)
  - Implementation/structure (“white box”)
- Do we execute things?
  - Yes: dynamic
  - No: review, analysis

# Unit testing

- Isolation of small testable components
- JUnit
- [also mentioned FIT]

# Testing in the Lifecycle

- Failures are common and persistent in large software projects.
- Bugs are really expensive to fix if we don't catch them early.
- Different lifecycles treat testing differently.

# Specification-based testing

- Random or systematic testing?
- Category-partition method
  - ITFs, parameters, environment
  - Categories
  - Constraints (reduce combinations)
  - Specification
  - Implement test cases
  - Execute
  - Evaluate

# Models

- All kinds:
  - Decision trees
  - Workflows
  - Finite state machines
  - Grammars
  - Flow between modal dialogs in GUIs
- Uses:
  - Manage randomized testing
  - Framework for measuring **coverage**
  - Method for reducing number of tests



# Structural testing: control flow

- Control Flow Graph: basic blocks and edges
- Coverage and **adequacy**:
  - Statement
  - Branch
  - Condition
  - Basic/compound condition, MC/DC
  - Path, loop interior boundary
- **Subsumption!**

# Structural testing: data flow

- Programs process data, so what happens to the data?
  - Graph/vertex/edge terminology
  - (Global) defs, computational uses, predicate uses
  - Def-clear paths
- Coverage:
  - All-defs
  - All-p-uses, all-c-uses, all/some
  - All-uses,
  - All-du-paths
  - All-paths
- Subsumption!

# Mutation testing

- Small variations to code
- Modelled on small programmer errors in software development
- Assumes:
  - Modelled **representatively** on human defects
  - Program is “close” to correct (competent programmer hypothesis)
  - Coupling effect hypothesis — tests good for small faults will also be good for large ones
- Measure test suite quality
- Measure residual defect density

# Integration testing

- Isolation of components hides component interactions
- So: systematically test interactions by **integration**
- Incrementally: top-down/bottom-up
- Can be laborious
  - Do we re-execute them *all*?
- Adequacy: coupling-based coverage
  - All-coupling-defs
  - All-coupling-uses
  - etc.

# Regression testing

- Software evolves
- “Fixes” don’t always fix the bug
- Many fixes introduce new bugs
- So re-use old tests?
- Minefield vs cloud analogies
- Which ones, how often, etc.
- Maintenance an issue
- Tool for managing change

# GUI testing

- Lots of different things to consider:
  - Usability, intuitiveness, guideline compliance, ...
- Used to be very laborious
- Can apply coverage again

## System & higher level testing

- Capacity
- Stress
- Usability
- Security
- Documentation
- Performance
- Reliability
- Availability
- Compliance
- Configuration

...and think how this all fits into the lifecycle

# What/how to test

- Functional and non-functional requirements
- Systematically vs randomly
- What's the right answer? (Oracle)
- Software lifecycle



# Test quality

- Coverage
  - Test “inadequacy”
  - Subsumption: better tests?
  - Control flow
  - Models
  - Anything which allows you to map the Software Under Test
- Mutation testing

# Exam

- Same format as in recent years: 2 of 3 questions.
- Revise recent exams!
- Pay attention to the question
  - Make sure that you identify everything that's asked.
- Manage your time
  - Pay attention to mark distribution within questions.
- (Pretend) I'm an idiot
  - Explain everything you do.