

# SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2017

## 35 Analysis (Ch. 6) DRAFT

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 19, 2017



# Analysis

- Administrative details
- The Oracle problem
- Module testing
- Integration testing
- Testing OO programs
- Testing concurrent and real-time systems
- Analysis
  - ▶ Code walk throughs and inspections

# Administrative Details

- Today's slides are partially based on slides by Dr. Wassong

# The Oracle Problem

- Given input test cases that cover the domain, what are the expected outputs?
- Oracles are required at each stage of testing to tell us what the right answer is
- Black-box criteria are better than white-box for building test oracles
- Automated test oracles are required for running large amounts of tests
- Oracles are difficult to design - no universal recipe

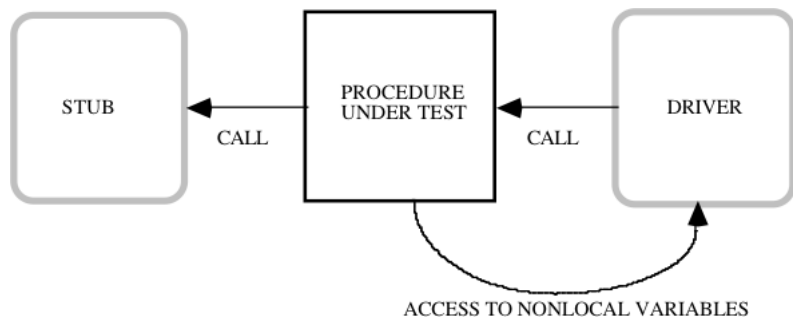
# The Oracle Problem Continued

- Determining what the right answer should be is not always easy
  - ▶ Air traffic control system
  - ▶ Scientific computing
  - ▶ Parallel testing can approximate an oracle
  - ▶ Properties of the expected values can be easier than stating the expected output

# Module Testing

- Scaffolding needed to create the environment in which the module should be tested
- Stubs - a module used by the module under test
- Driver - module activating the module under test

# Testing a Functional Module



# Integration Testing

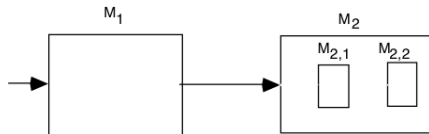
- Big-bang approach
  - ▶ First test individual modules in isolation
  - ▶ Then test integrated system
- Incremental approach
  - ▶ Modules are progressively integrated and tested
  - ▶ Can proceed both top-down and bottom-up according to the USES relation



# Integration Testing and USES relation

- If integration and test proceed bottom-up only need drivers
- Otherwise if we proceed top-down only stubs are needed

# Example

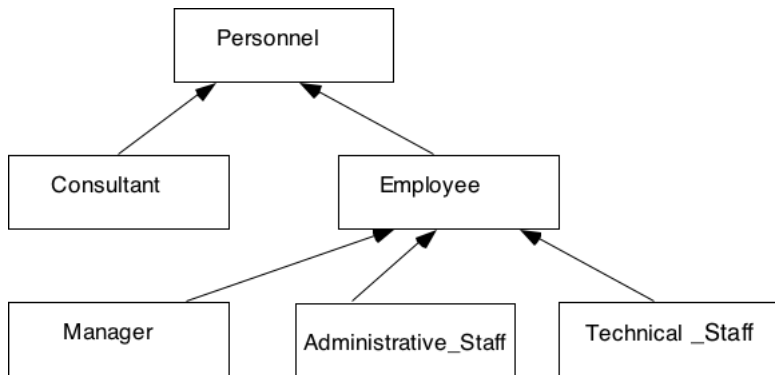


- $M_1$  USES  $M_2$  and  $M_2$  IS\_COMPOSED\_OF  $\{M_{2,1}, M_{2,2}\}$
- Case 1
  - ▶ Test  $M_1$  providing a stub for  $M_2$  and a driver for  $M_1$
  - ▶ Then provide an implementation for  $M_{2,1}$  and a stub for  $M_{2,2}$
- Case 2
  - ▶ Implement  $M_{2,2}$  and test it by using a driver
  - ▶ Implement  $M_{2,1}$  and test the combination of  $M_{2,1}$  and  $M_{2,2}$  (i.e.  $M_2$ ) by using a driver
  - ▶ Finally implement  $M_1$  and test it with  $M_2$  using a driver for  $M_1$

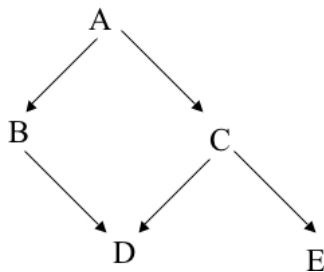
# Testing OO Programs

- New issues
  - ▶ Inheritance
  - ▶ Genericity
  - ▶ Polymorphism
  - ▶ Dynamic binding
- Open problems still exist

# Inheritance



# How to Test Classes of the Hierarchy



- “Flattening” the whole hierarchy and considering every class as totally independent component
- This does not exploit incrementality
- Finding an ad-hoc way to take advantage of the hierarchy
- Think about testing `PointT.java` and `PointMassT.java`

# A Sample Strategy

- A test that does not have to be repeated for any heir
- A test that must be performed for heir class X and all of its further heirs
- A test that must be redone by applying the same input data, but verifying that the output is not (or is) changed
- A test that must be modified by adding other input parameters and verifying that the the output changes accordingly

# Testing Concurrent and Real-time Systems

- Nondeterminism inherent in concurrency affects repeatability
- For real-time systems, a test case consists not only of input data, but also of the times when such data are supplied
- Considerable care and detail when testing real-time systems

# Analysis Versus Testing

- Testing characterizes a **single** execution
- Analysis characterizes a **class** of executions; it is based on a **model**
- They have complementary advantages and disadvantages



# Informal Analysis Techniques and Code Walkthroughs

- Recommended prescriptions
  - ▶ Small number of people (three to five)
  - ▶ Participants receive written documentation from the designer a few days before the meeting
  - ▶ Predefined duration of the meeting (a few hours)
  - ▶ Focus on the **discovery** of errors, not on fixing them
  - ▶ Participants: designer, moderator, and a secretary
  - ▶ Foster cooperation; no evaluation of people
  - ▶ Experience shows that most errors are discovered by the designer during the presentation, while trying to explain the design to other people

# Informal Analysis Techniques Code Inspection

- A reading technique aiming at error discovery
- Based on checklists
  - ▶ Use of uninitialized variables
  - ▶ Jumps into loops
  - ▶ Nonterminating loops
  - ▶ Array indexes out of bounds

# Correctness Proofs

- Formal program analysis is a verification aid that may enhance program reliability
- Mathematically prove that the program's semantics implies its specification
- Can use pre and post conditions
- Tabular expressions can be proven to match between specification of requirements and a specification of the design
- In many cases verification can be automated, at least partially

# Model Checking

- Correctness verification, in general, is an undecidable problem
- Model checking is a rather recent verification technique based on the fact that most interesting system properties become decidable (algorithmically verifiable) when the system is modelled as a finite state machine

# Model Checking Continued

- Describe a given system - software or otherwise - as an FSM
- Express a given property of interest as a suitable formula
  - ▶ Does a computation exist that allows a process to enter a critical region?
  - ▶ Is there a guarantee that a process can access shared resources?
- Verify whether the system's behaviour does indeed satisfy the desired property
  - ▶ This step can be performed automatically
  - ▶ The model checker either provides a **proof** that the property holds or gives a **counter example** in the form of a test case that exposes the system's failure to behave according to the property

# Why so Many Approaches to Testing and Analysis?

- Testing versus (correctness) analysis
- Formal versus informal techniques
- White-box versus black-box techniques
- Techniques in the small/large
- Fully automatic versus semi-automatic techniques (for undecidable problems)
- ...

View all of these as complementary

# Debugging

- The activity of locating and correcting errors
- It can start once a failure has been detected
- The goal is closing the gap between a fault and a failure
  - ▶ Memory dumps, watch points
  - ▶ Intermediate assertions can help
  - ▶ Tools like gdb, valgrind, etc.