Interaction vs State Testing

Carl Erickson
Atomic Object LLC

Reference: “Mock Roles, Not Objects”
Freeman, Price, Mackinnon, Walnes,
OOPSLA 2004
Background

• Conventional unit tests invoke a method then make assertions about the state of the test object
• TDD is a design activity that changes programming from invention to discovery
• McKinnon, Freeman, Craig
  – Mock Objects, 2001
  – Better structured tests, “interface discovery”
Lean Inspiration

• Pull is favored over push, as pull arises from demand, whereby push arises from forecast
• For an object’s interface, the “pull” is the demand by the clients of the object
• Need-driven development is the process of discovery these demands using Mock Objects
Need-driven development

• Devise a test for object A
• To satisfy this test uncovers a need for service S provided by some other object

- Create a mock object to provide S so you can keep focus on developing A
Example, cont.

- Once A is completed, switch focus to an object B that satisfies service S
- Tests uncover need in B for T and U
mock.expect(expectation)
  .method(method name)
  .with(argument)
  .after(id of prior invocation)
  .match(other matching rule)
  .will(stubbed behavior)
  .id(id of this invocation);
End Result

• A network of objects which
  – follow Demeter’s Law
  – satisfy requirements per test spec
  – communicate through minimal, complete interfaces
    (and hence require least amount of code to implement)

• A design which
  – favors composition over inheritance

• Tests which
  – confirms interaction between objects, not their state
Advise & Experience

• Mock only types you own
  – If you can’t change it, don’t mock it (wrap it instead)
  – Emphasizes Mock Objects as a design activity

• Don’t use getters
  – Leads to tighter coupling by exposing impl
  – Can cause responsibility to be misplaced

• Specify what should not happen
Advise & Experience, cont.

• Specify as little as possible
  – Makes for brittle tests

• Don’t mock boundary objects
  – No need to mock if they don’t use other objects
  – Assert that they return proper values (state-based)

• Too many mocks
  – Indicates bad design

• Use dependency injection
  – Pass objects in via parameter or constructor, or use a factory
JMock

- A library to support need-driven development
  - Easily create mock classes
  - Specify expected interactions (invocations of methods)
  - Stub invoked behavior