System Testing

Emina Torlak
emina@cs.washington.edu
Outline

• Recap: system testing
• Integration testing
• Performance testing
recap

system testing
System testing

- **System testing**: tests the behavior of a system as a whole, with respect to scenarios and requirements
  - Functional testing, integration testing
  - Load, stress, performance testing
  - Acceptance, usability, installation, beta testing
System testing

- **System testing**: tests the behavior of a system as a whole, with respect to scenarios and requirements
  - Functional testing, integration testing
  - Load, stress, performance testing
  - Acceptance, usability, installation, beta testing
integration testing
Integration testing
Integration testing

- **Integration testing:** checking software quality by testing two or more dependent software modules as a group or a (sub)system.
Integration testing

- **Integration testing**: checking software quality by testing two or more dependent software modules as a group or a (sub)system.

- Challenges same as in unit testing, plus:
  - Combined units can fail in more places and in more complicated ways.
  - How to test a partial system where not all parts exist?
  - How to "rig" the behavior of unit A so as to produce a given behavior from unit B?
Stubs: a way to test a partial system
Stubs: a way to test a partial system

- **Stub**: A controllable replacement for an existing software unit to which your code under test has a dependency.
Stubs: a way to test a partial system

- **Stub**: A controllable replacement for an existing software unit to which your code under test has a dependency.

- Useful for simulating difficult-to-control elements:
  - network / internet
  - time/date-sensitive code
  - database, files, io, threads, memory
  - brittle legacy code /systems
Testing with stubs (1/3)

- Identify the external dependency.
  - This is either a resource or a class/object.
  - If it isn't an object, wrap it up into one.
  - (Suppose that Class A depends on Class B.)
Testing with stubs (2/3)

• Extract the core functionality of the object into an interface.
  • Create an InterfaceB based on B
  • Change all of A's code to work with type InterfaceB, not B
Testing with stubs (3/3)

• Write a second "stub" class that also implements the interface, but returns pre-determined fake data.
  • Now A's dependency on B is abstracted away and can be tested easily.
  • Can focus on how well A integrates with B's external behavior.
Where to inject stubs?
Where to inject stubs?

- **Seams**: places to inject the stub so Class A will talk to it.
  
  - at construction (not ideal)
    ```java
    A aardvark = new A(new StubB());
    ```
  
  - through a getter/setter method (better)
    ```java
    A apple = new A(...);
    aardvark.setResource(new StubB());
    ```
  
  - just before usage, as a parameter (also better)
    ```java
    aardvark.methodThatUsesB(new StubB());
    ```
Where to inject stubs?

• **Seams**: places to inject the stub so Class A will talk to it.
  
  • at construction (not ideal)
    
    A aardvark = new A(new StubB());
  
  • through a getter/setter method (better)
    
    A apple = new A(...);
    aardvark.setResource(new StubB());
  
  • just before usage, as a parameter (also better)
    
    aardvark.methodThatUsesB(new StubB());
  
• You should not have to change A's code everywhere (beyond using your interface) in order to use your Stub B. (a "testable design")
Mock objects: a way to test interactions

• **Mock object**: A fake object that decides whether a unit test has passed or failed by watching interactions between objects.

• Useful for **interaction** testing, as opposed to **state** testing
Stubs vs mocks

- A stub (B) gives out data that goes to the object/class under test (A).
  - The unit test directly asserts against A, to make sure it gives the right result when fed B’s data.
Stubs vs mocks

• A stub (B) gives out data that goes to the object/class under test (A).
  • The unit test directly asserts against A, to make sure it gives the right result when fed B’s data.

• A mock (B) waits to be called by the class under test (A).
  • It may have several methods it expects that A should call.
  • It makes sure that it was called in exactly the right way.
  • If A interacts with B the way it should, the test passes.
Mock object frameworks

• Stubs are often best created by hand/IDE.Mocks are tedious to create manually.

• Mock object frameworks help with the process.
  • android-mock, EasyMock, jMock (Java)
  • ...

• Frameworks provide the following:
  • auto-generation of mock objects that implement a given interface
  • logging of what calls are performed on the mock objects
  • methods/primitives for declaring and asserting your expectations
Mock object frameworks

• Stubs are often best created by hand/IDE. Mocks are tedious to create manually.

• Mock object frameworks help with the process.
  • android-mock, EasyMock, jMock (Java)
  • ...

• Frameworks provide the following:
  • auto-generation of mock objects that implement a given interface
  • logging of what calls are performed on the mock objects
  • methods/primitives for declaring and asserting your expectations
import org.jmock.integration.junit4.*; // Assumes that we are testing
import org.jmock.*; // class A's calls on B.

@RunWith(JMock.class)
public class ClassATest {
    private Mockery mockery = new JUnit4Mockery(); // initialize jMock

    @Test  public void testACallsBProperly1() {
        // create mock object to mock InterfaceB
        final InterfaceB mockB = mockery.mock(InterfaceB.class);

        // construct object from class under test; attach to mock
        A aardvark = new A(...);
        aardvark.setResource(mockB);

        // declare expectations for how mock should be used
        mockery.checking(new Expectations() {
            oneOf(mockB).method1("an expected parameter");
            will(returnValue(0.0));
            oneOf(mockB).method2();
        });

        // execute code A under test; should lead to calls on mockB
        aardvark.methodThatUsesB();

        // assert that A behaved as expected
        mockery.assertIsSatisfied();
    }
}
Using stubs and mocks together
Using stubs and mocks together

• Suppose a log analyzer reads from a web service. If the web fails to log an error, the analyzer must send email.
Using stubs and mocks together

• Suppose a log analyzer reads from a web service. If the web fails to log an error, the analyzer must send email.

• How to test to ensure that this behavior is occurring?
  • Set up a stub for the web service that intentionally fails.
  • Set up a mock for the email service that checks to see whether the analyzer contacts it to send an email message.
performance testing
Acceptance, performance

- **Acceptance testing**: System is shown to the user / client / customer to make sure that it meets their needs.
  - A form of black-box system testing.

- Performance is important.
  - Performance is a major aspect of program acceptance by users.
  - Your intuition about what's slow is often wrong.
Acceptance, performance

• **Acceptance testing**: System is shown to the user / client / customer to make sure that it meets their needs.
  
  • A form of black-box system testing.

• Performance is important.
  
  • Performance is a major aspect of program acceptance by users.
  
  • Your intuition about what's slow is often wrong.

Premature optimization is the root of all evil.

*Donald Knuth*
Thinking about performance
Thinking about performance

• The app is only too slow if it doesn't meet your project’s stated performance requirements.
  • If it meets them, DON'T optimize it!
Thinking about performance

• The app is only too slow if it doesn't meet your project’s stated performance requirements.
  • If it meets them, DON'T optimize it!

• Which is more important, fast code or correct code?
Thinking about performance

• The app is only too slow if it doesn't meet your project’s stated performance requirements.
  • If it meets them, DON'T optimize it!

• Which is more important, fast code or correct code?

• What are reasonable performance requirements?
  • What are the user's expectations? How slow is "acceptable" for this portion of the application?
  • How long do users wait for a web page to load?
  • Some tasks (admin updates database) can take longer
Profile and measure before optimizing

• Runtime / CPU usage
  • what lines of code the program is spending the most time in
  • what call/invocation paths were used to get to these lines

• Memory usage
  • what kinds of objects are on the heap
  • where were they allocated
  • who is pointing to them now
  • "memory leaks" (does Java have these?)

• Web page load times, requests/minute, …
Profile and measure before optimizing

- Runtime / CPU usage
  - what lines of code the program is spending the most time in
  - what call/invocation paths were used to get to these lines

- Memory usage
  - what kinds of objects are on the heap
  - where were they allocated
  - who is pointing to them now
  - "memory leaks" (does Java have these?)

- Web page load times, requests/minute, …

CPU profiling slows down your code (a lot). Design your profiling tests to be very short.
Optimization hints: think high-level
Optimization hints: think high-level

• Focus on high-level optimizations (algorithms, data structures)
  • Leave the low-level ones to the compiler
Optimization hints: think high-level

• Focus on high-level optimizations (algorithms, data structures)
  • Leave the low-level ones to the compiler

• Some common high-level optimizations
  • **Lazy evaluation** saves you from computing/loading
    • don't read / compute things until you need them
  • **Hashing, caching** save you from reloading resources
    • combine multiple database queries into one query
    • save I/O / query results in memory for later Web page load times, requests/minute, etc.
  • **Precomputing** values and storing them in a lookup table
    • the first 1000 primes
Summary

• System testing checks the behavior of a system as a whole.

• Integration testing checks software quality by testing two or more dependent software modules as a group.

• Performance testing checks that a system meets performance requirements (e.g., responsiveness).