Software Design (F28SD2)
Dynamic Analysis Techniques
Part 2

Andrew Ireland
Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Review

- Functional testing:
  - Equivalence partitioning
  - Boundary value analysis
- Structural testing:
  - Code coverage metrics, i.e. statement, decision, condition/decision, multiple condition, MC/DC.
- Hybrid testing:
  - Error guessing
  - Assertion based testing

Error Guessing

- Error guessing is driven by experience and will typically cover both functional and structural perspectives.
- To illustrate, given a software component to test that exploits dynamic storage allocation, then an experienced tester will typically focus upon the correct use of resource deallocation, *e.g.* seek out memory leakage defects.
- By its very nature, error guessing is hard to teach. However, check lists designed around language features and related defects can play a useful role in the transfer of experience.

Assertion Based Testing

- An **assertion** is a boolean valued expression, *i.e.* evaluates to `true` or `false`.
- *"It may be true, that whenever [control] actually reaches a certain point in the flow diagram, one or more bound variables will necessarily possess certain specified values, or possess certain properties, or satisfy certain relations with each other."* **Assertion Boxes**, Goldstine & von Neuman 1947
- Assertions allow a programmer to express what they believe **should** be true at given points within their code.
- While we only consider assertions here within the context of dynamic analysis, *i.e.* assertions that are executed, in the next lecture we will highlight their broader use within software verification.
- Assertions will be represented differently depending upon your implementation language – we will focus upon assertions within Java.

Assertion Based Object Oriented Testing

- **Encapsulation:** gaining access to states and hidden instance variables is crucial if effective testing is to be conducted.
- **Inheritance and polymorphism:** overriding does not mean local testing is enough. That is, an inherited method that has be fully tested within the superclass must be re-tested within the context of the subclass.
- **Unit testing**: treating methods as the basis for unit testing will not work in general, a class perspective is called for, *i.e.* class states, class modality etc.

Design by Contract

- Strong analogy with legal contracts, *i.e.* an explicit statement about the benefits (rights) and obligations that exist between a client and supplier.
- Each supplier method has an associated precondition assertion that checks the correctness of client requests – client responsible for establishing precondition.
- Each supplier method has an associated postcondition assertion that checks the correctness of the supplier's:
  - response to the client and
  - suppliers state after the response is calculated.

  supplier responsible for establishing postcondition.

Note: intermediate method assertions may also be desirable, *e.g.* loop invariants.

Obligations and Benefits

	Obligations	**Benefits**
Client	Must ensure preconditions, *e.g. Ensure space in table for given key/value pair*	May benefit from postconditions, *e.g. Obtain new table updated with given key/value pair*
Supplier	Must ensure postconditions, *e.g. Ensure given key/value pair has been correctly added to the table*	May assume preconditions, *e.g. No need to consider if table is full*

Java Support for Assertions

- J2SE (Java 2 Platform, Standard Edition) 1.4 and above supports a simple assertion facility:
  - new keyword, *i.e.* `assert`
  - new class, *i.e.* `AssertionError`
  - some additional methods added to `java.lang.ClassLoader`
- J2SE 1.7 is the default on the departmental Linux PCs.

Java Assertions: Syntax & Semantics

Syntax: `assert` statements come in 2 forms:

- `assert expression1;`
- `assert expression1 : expression2;`

where `expression1` and `expression2` must be of type `boolean` and `String` respectively.

Semantics: execution requires evaluation of `expression1`:

- If `expression1` evaluates to `true` then no effect.
- If `expression1` evaluates to `false` then
  - default `AssertionError` constructor invoked (if no `expression2`).
  - otherwise evaluate `expression2` and use result with single-parameter `AssertionError` constructor.

A Simple Example

```
class Foo
{
  public static void even(int value)
  {
      assert 0 <= value;
      if (value % 2 == 0) System.out.println("Even");
                     else System.out.println("Odd");
  }
  public static void main(String[] args)
                                      throws IOException
  {   even(2);
      even(3);
      even(-1);
  }
}
```

Compilation and Execution

- By default the Java run-time system ignores assertions, so again a command line option to enable assertions is required, *e.g.*

  ```
  java -enableassertions Foo
  ```

  fortunately this can be abbreviated to:

  ```
  java -ea Foo
  ```

- Note that default behaviour a little backward, developers are being very cautious.

Example Revisited

- **Compile:**

```
$ javac Foo.java
```

- **Execute with assertions disabled:**

```
$ java  Foo
Even
Odd
Odd
```

- **Execute with assertions enabled:**

```
$ java -ea Foo
Even
Odd
Exception in thread "main" java.lang.AssertionError
        at Foo.even(Foo.java:12)
        at Foo.main(Foo.java:23)
```

Assertion Messages

```
class Foo
{
  public static void even(int value)
  {
    assert 0 <= value : "Value must not be negative:
                         value= " + value;
    ...
  }
  ...

  $ javac Foo.java
  $ java -ea Foo
  Even
  Odd
  Exception in thread "main" java.lang.AssertionError:
  Value must not be negative: value= -1
          at Foo.even(Foo.java:12)
          at Foo.main(Foo.java:23)
```

A More Interesting Example

```
class Book
{
    class Entry
    {
        String key;
        String value;
        Entry(String key, String value)
        {
            this.key = key;
            this.value = value;
        }
    }
...
```

A More Interesting Example

```
...
    private int capacity;
    private int index;
    private Entry [] book;

    public Book(int capacity)
    {
        this.book = new Entry [capacity];
        this.capacity = capacity;
        this.index = 0;
    }
...
```

Specification via Comments

```
public void insertData(String key, String data)
{
  // assume data count is less than capacity

  ... your array insertion algorithm goes here ...

  // book now contains given data and
  // is associated with the given key and
  // data count has been increased by 1

}
```

Specification via Assertions

```
public void insertData(String key, String data)
{
    int initialCount = count();

    assert initialCount < capacity

  ... your array insertion algorithm goes here ...

    assert contains(data) &&
           iskey(data) == key &&
           count() == initialCount + 1
}
```

Specification & Implementation

```
public void insertData(String key, String data)
{
    int initialCount = count();

    assert initialCount < capacity

    Entry e = new Entry(key, data);
    book[index] = e;
    index++;

    assert contains(data) &&
           iskey(data) == key &&
           count() == initialCount + 1
}
```

Auxiliary Methods

```
boolean contains(String data)
    {
        int i=0;
        boolean found = false;
        while (i < index && !found){
            if (book[i].value == data) found = true;
            else i++;
        }
        return found;
    }
```

Exercise: Define the `count` and `iskey` methods.

Class Invariant for Book

```
boolean BookClassInvariant()
{
    int cnt = count();
    return cnt >= 0 && cnt <= capacity;
}
public Book(int capacity)
{
    ... code ...
    assert BookClassInvariant()
}
public void insertData(String key, String data)
{
    assert BookClassInvariant()
    ... code ...
    assert BookClassInvariant()
}
```

Design by Contract (more)

- Each class has an associated invariant assertion that characterizes the space of correct instance variable values for the class. Invariant assertions are checked when method precondition and postcondition assertions are checked.
- Class contracts must be consistent across subclass/superclass definitions, *i.e.* a subcontractor (supplier) that is employed to perform a specialized task takes on part of the responsibility of the main contractor (supplier):
  - Preconditions of subclass method must be the same (or weaker) than the associated superclass method.
  - Postconditions of subclass method must be the same (or stronger) than the associated superclass method.
  - Subclass invariant must be the same (or stronger) than the superclass invariant.

Contracts, Inheritance and Robustness

- Contract consistency across subclass/superclass definitions is not mechanized within Java, *i.e.* needs to be checked off-line by hand.
- The **Eiffel** object-oriented programming language pioneered the notion of **design-by-contract**, using assertions called **require** (precondition) and **ensure** (postcondition) – for more details see `http://www.eiffel.com/`
- Input validation checks should not be confused with assertions, *i.e.*
  - Input validation checks are about **robustness**
  - Assertions are about **correctness**
- Input validation checks and assertions are complementary.

JUnit: A Framework for Unit Testing

- JUnit provides a unit testing framework for Java applications.
- Users write test cases, *i.e.* inputs and expected results.
- JUnit automates the execution and presentation of the results.
- Typically promoted in the eXtreme programming community, *i.e.* testing and coding should go hand-in-hand, with test case generation leading the way.
- Standalone: Text (`textui`); AWT (`awtui`); Swing (`swingui`).
- IDE: Eclipse; JBuilder; ...
- The basic approach has be adopted by others, *e.g.* PHP (PHPUnit), Python (PyUnit), C++ (CPPUnit), ...

What is a JUnit Test?

- Unit tests are represented by test classes, and can be viewed as "drivers".
- For example, consider a `Book` class:
  - Class `Book` has `new`, `insertData`, `lookupData`, ...
  - Class `TestBook` has `testEmptyBook`, `testAddOneBook`, `testIntegrityBook`
- Unit tests are expressed in terms of **assertions**, *i.e.* `assertTrue`, `assertFalse`, `assertNull`, `assertNotNull`, `assertEquals`, `assertSame`, `assertNotSame`, ...
- For a complete list of the JUnit assertions see: `http://www.junit.org/apidocs/org/junit/Assert.html`

JUnit Testing of the Book Class

Consider some tests:

- A new Book has zero entries.
- A new Book which has been updated with one entry, has only one entry!
- Given an entry with key X and value Y, then looking up X will retrieve Y.

Now let's see how to implement them as JUnit tests ...

A Test Case Class

```
import junit.framework.*;
import junit.extensions.*;

public class BookTest extends TestCase
{
    public void testEmptyBook()
    {
        assertEquals(0, new Book(3).index);
    }
    ...
}
```

Note that `TestCase` is a JUnit class that you extend. Note also that package `junit.framework` contains all the basic JUnit classes, while `junit.extensions` contains experimental extensions to JUnit.

A Test Case Class [more]

```
...
public class BookTest extends TestCase
{   ...
    public void testAddOneBook()
    {
        Book bk = new Book(1);
        bk.insertData("Micky", "Disney");
        assertEquals(1, bk.index);
    }
    public void testIngerityPosBook()
    {
        Book bk = new Book(1);
        bk.insertData("Micky", "Disney");
        assertEquals("address-name missmatch", "Disney",
        bk.lookupData("Micky")); }
}
```

Running JUnit Tests [partial success]

```
java -cp .:/usr/local/java/bluej/lib/junit-4.8.2.jar \
      junit.textui.TestRunner                        \
      BookTest
..F.
Time: 0.004
There was 1 failure: 1)
testIngerityPosBook(BookTest)
junit.framework.ComparisonFailure:
address-name missmatch expected:<[Disney]>
                          but was:<[Failure]>

...

FAILURES!!!
Tests run: 3,  Failures: 1,  Errors: 0
```

Note: cp = class search path for directories and zip/jar files.

lookupData [buggy]

```
public String lookupData(String k)
{
    int i=0;
    boolean found = false;
    while (i < index && !found){
        if (book[i].value == k) found = true;
        else i++;
    }
    if (found) return book[i].value;
    else return "Failure";
}
```

Note: `book[i].value` returns that `value` associated with entry `i`, and not the `key`.

lookupData [patched]

```
public String lookupData(String k)
{
    int i=0;
    boolean found = false;
    while (i < index && !found){
        if (book[i].key == k) found = true;
        else i++;
    }
    if (found) return book[i].value;
    else return "Failure";
}
```

Running JUnit Tests [success]

```
java -cp .:/usr/local/java/bluej/lib/junit-4.8.2.jar \
      junit.textui.TestRunner \
      BookTest
...
Time: 0.002

OK (3 tests)
```

`setUp` and `tearDown`

- Typically a family of tests will work with a common set of objects.
- But each test will potentially change some or all of the objects – so how can we minimize the work involved in constructing such test objects whilst ensuring tests do not interfere with each other?
- JUnit has a simple but effective answer:
  - `setUp()` allows you to create objects that will be available when each unit test is run.
  - `tearDown()` undoes the work of `setUp()`.

Note: `setUp()` and `tearDown()` are defined within your test class.

Summary

- The importance of error guessing.
- The value of executable assertions – “active comments”.
- Assertions within the context of object-oriented programming and Java in particular.
- Design by Contract.
- Unit testing with assertions via JUnit.
- Note: Assertions can be buggy!

References

- "Testing Object-Oriented Systems", Binder, R.V. Addison-Wesley, 1999.
- "Design By Contract", Meyer, B. IEEE Computer 25(10):40-51, 1992.
- "JUnit: Pocket Guide", Beck, K. O'Reilly, 2004.
- "Software Testing", Hambling, B. (Editor), British Computer Society Publication, 2008.

References

- Java assertions – `http://java.sun.com/developer/technicalArticles/JavaLP/assertions/`
- JUnit – `http://www.junit.org`
- AssertMate – a code assertion system for Java, RST Corp: `http://www.cigital.com/products/`
- Test Coverage Tool Venders: `http://www.iplbath.com` `http://www.reflex-tech.co.uk`