CITS5501 Software Testing and Quality Assurance
Introduction to testing; unit testing

Unit coordinator: Arran Stewart

Highlights

- Testing concepts
- Documenting code
- APIs
- Unit testing
- Testable documentation
- Property-based testing

Highlights

- Documentation and APIs: how do we work out what the correct behaviour of a piece of software *is* so that we can test it?
- Unit testing: What is unit testing, what is the terminology, and how do we write unit tests?

Testing concepts (cont'd)

Faults, failures and errors

Last lecture, we were talking about *faults*, *failures* and *erroneous states*.

In normal English, we might not make much distinction between them.

But in software engineering, it can be useful to distinguish whether we're talking about

- the *behaviour* of a running system (what we can observe about what the system does)
- the *static artifacts* from which the system is produced (e.g. files of source code, or data files in formats like HTML or CSS)
- the *runtime state* of the system (i.e., what's currently stored in memory)

(Aside: terminology)

Note that we will use terminology that's consistent with:

- Ammann & Offutt,[1] and
- Bruegge & Dutoit[2]

but you may find other sources that use different terminology.

(In particular, sources vary greatly in what they consider "error" to mean.)

[1] Amman, P. & J. Offut, *Introduction to Software Testing* (2nd edn, 2017)

[2] Dutoit, B.B. and A.H. Dutoit., *Object-Oriented Software Engineering: Using UML, Patterns, and Java* (3rd edn, 2014).

Failures

Recall that:

- A *failure* is any deviation of the observed behaviour of a program or system from the specification.

It describes the system's *behaviour*.

(Can we say a file of source code contains a "failure"? No. But can we say a failure occurs when some program is run? Yes.)

Failure examples

If a program should save a document when the user types "`ctrl-s`", but instead crashes when the filename contains a space – that would be a failure.

- The program would be failing to meet a *functional* requirement

If a program should always respond to user input within 0.1 seconds, but instead starts "lagging" when more than 5 documents are open – that's also a failure.

- This time, the program would be failing to meet a *non*-functional requirement, that the system meet particular standards for *responsiveness*.

If an electronic voting booth should accurately record votes cast, but due to a cosmic ray flipping a bit in memory, 4096 additional votes are counted for one candidate – that's also a failure.

Faults

Also called *bugs* or *defects*.

- A *fault* is something in the static artifacts of a system that causes a failure.

Faults

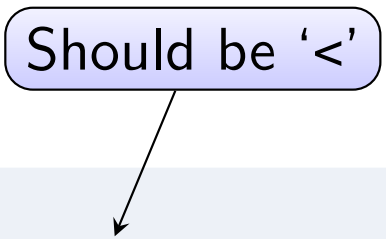
Also called *bugs* or *defects*.

- A *fault* is something in the static artifacts of a system that causes a failure.

For an example of a fault, consider the following Java code, intended to iterate over an array of book titles and print them out:

```
for(int i = 0; i <= book_titles.length; i++) {
  System.out.println(book_titles[i]);
}
```

Fault example




```
for(int i = 0; i <= booktitles.length; i++) {
  System.out.println(booktitles[i]);
}
```

Broad definition of "fault"

Some sources will use "fault" more broadly to mean the *cause* of a failure besides just defects in the code – e.g. perhaps cosmic rays – but we'll mostly be concerned with problems in the code.

Not every failure can be traced back to a *single* spot in the code: failures of security, scalability, performances etc. are global properties of the system artifacts.

Erroneous state

By "error" or "erroneous state", the textbook authors mean the situation at runtime, where some fault has become reflected in the system's runtime state.

- So you can have a *fault* in the code (e.g. off-by-one Java loop error we saw), but if we execute the program and (at least this time round) that bit of code doesn't happen to get run, we don't get a corresponding erroneous state.

Invariants

We will mostly be interested in erroneous states that happen because some *class* or *program invariant* has been violated.

Class invariant example – a stack

```
/** A Stack data type, implemented using an array */
class ArrayStack implements Stack {
  int elements[];
  /* class invariant: topOfStack always points to current top item
   * (or -1 if empty)*/
  int topOfStack;
  public push(int value) {
     topOfStack += 1;
     if (topOfStack >= 20) { /* throw exception */ }
     elements[topOfStack] = value;
  }
  public int pop() {
     if (topOfStack < 0) { /* throw exception */ }
     return elements[topOfStack];
  }
  // ... other methods ...
}
```

topOfStack is never decremented!

System invariant example – databases

Suppose our system has a database, with records representing *students*, *units*, and *enrollments*.

An enrollment is a `(studentId, unitId)` pair, e.g.: `(23456789, CITS5501)`.

It's a (plainly sensible) rule of our system that an enrollment record must not contain a student ID for a student that doesn't exist, nor a unit ID for a unit that doesn't exist.

If this rule is breached (perhaps a unit gets removed, and the corresponding enrolments don't), then our system is now in an inconsistent (or erroneous) state.

Invariants

We will look at invariants more in the lab/workshops.

Reliability

- The *reliability* of a system is the degree to which its observed behaviour conforms to its specification.

Testing

Definition

We define testing as a systematic attempt to find faults in a planned way a software system.

(Adapted from Bruegge & Dutoit)

What sorts of things can we test?

We can classify tests by the "level" of component or system they work with:

- We can test a single procedure, method or function – this is called *unit testing*. A *unit* just means a component of code – typically a small one, but sometimes the term is used to refer to modules (collections of definitions).
- We can test how units, classes, modules or other components of a system work together – this is called *integration testing*
- We can test an entire system – this is called *system testing*

What sorts of things can we test?

We can also classify them by the purpose of the test, or when in the software development lifecycle the testing activity occurs:

- After making a change to some component (an enhancement, or bug fix, or re-factoring): we can check whether it still passes all relevant tests. This is called *regression testing*.
- On delivery of a system: we can 'test' whether a system meets a customer's expectations – this is called *acceptance testing*

Testing

- Testing requires a different mind-set from construction: when constructing (or designing) software, we usually focus on what it will do when things go *right*;
  when testing, we focus on *finding faults* – occasions when things do wrong.
- Programmers do often refer to tests as "failing" – but when a test indicates a bug, that's actually example of it *succeeding* in its purpose (i.e., showing the presence of a fault)

Unit testing and unit specifications

- We'll start by looking at the "lowest" level of tests, unit tests.
- When we test a unit of code, we aim to (by finding faults and removing them) increase our confidence that it meets its specifications.
  If we don't *have* any specifications for it, that obviously makes life difficult.
- So in general, we aim to *document* the intended behaviour of any externally available unit.

  (Some units might be purely internal – "private", for instance, in Java – it is usually good practice to document those as well, if we're going to test them – else how will we know what to test for?)

Documenting code

Documenting units

- Most modern languages provide some way of documenting the specification of units *inline* (that is, in the body of the code, rather than in a separate reference manual) and extracting that documentation for use by developers.
- For instance:
  - Java provides the Javadoc tool
  - Python provides the Pydoc tool
- For languages which do not have such a tool, applications such as Doxygen allow units to be documented and the documentation extracted.

Javadoc example

Consider the task of finding the position of the first occurrence of some character in a string.

In Java, the `String.indexOf()` method will do this for us.

Its signature is:

```
int indexOf(int ch)
```

That is, it takes an `int` and returns an `int`. (Why not a `char`? For historical reasons we won't get into.)

Javadoc example, cont'd

If we look up the Java documentation for the `indexOf` method, we will get the following information:

> *Returns the index within this string of the first occurrence of the specified character. If a character with value ch occurs in the character sequence represented by this String object, then the index (in Unicode code units) of the first such occurrence is returned. For values of ch in the range from 0 to 0xFFFF (inclusive), this is the smallest value k such that:*
> `this.charAt(k) == ch`
> *is true. For other values of ch, it is the smallest value k such that:* `this.codePointAt(k) == ch` *is true. In either case, if no such character occurs in this string, then -1 is returned.*
>
> ***Parameters:***
> *ch - a character (Unicode code point).*
>
> ***Returns:***
> *the index of the first occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur.*

Javadoc example, cont'd

Key points from the Javadoc documentation:

When we call `someString.indexOf(ch)`:

- If `ch` is *not* in `someString`, the method returns `-1`
- If `ch` *is* in `someString`, the method returns "the smallest value `k` such that $>$ `this.codePointAt(k) == ch`"

  (Or: "The first position in `someString` where the character `ch` appears." Which of the two is easier to understand? Which is easier to write a test for?)

Javadoc example, cont'd

Key points from the Javadoc documentation:

When we call `someString.indexOf(ch)`:

- If `ch` is *not* in `someString`, the method returns -1
- If `ch` *is* in `someString`, the method returns "the smallest value `k` such that `this.codePointAt(k) == ch`"

  (Or: "The first position in `someString` where the character `ch` appears." Which of the two is easier to understand? Which is easier to write a test for?)

(Are there any other possibilities not covered by the documentation?)

Javadoc example, cont'd

The documentation for the `indexOf` method is produced from a specially written *comment* which looks something like this:

```
/** Returns the index within this string of the first occurrence of the
 * specified character. If a character with value ch occurs in the character
 * sequence represented by this String object, then the index (in Unicode code
 * units) of the first such occurrence is returned. For values of ch in the range
 * from 0 to 0xFFFF (inclusive), this is the smallest value <i>k</i> such that:
 * this.charAt(<i>k</i>) == ch
 * is true. For other values of ch, it is the smallest value <i>k</i> such that:
 * this.codePointAt(<i>k</i>) == ch
 * is true. In either case, if no such character occurs in this
 * string, then -1 is returned.
 *
 * @param   ch   a character (Unicode code point).
 * @return  the index of the first occurrence of the character in the
 *          character sequence represented by this object, or
 *          -1 if the character does not occur.
 */
```

Javadoc conventions

The Javadoc comment is normally placed just before the method it documents, and begins with a double asterisk ("/**")

- It describes what the method does, what parameters should be passed in, and what result will be returned.
- It uses the `@param` markup to describe each parameter.
- It uses the `@return` markup to describe the return value.

Pydoc

In Python, the nearest equivalent method to Java's `indexOf` would be `String.index`, which searches for a substring within another string.

It does not actually have Pydoc documentation, but if it did, it would look like this:

```
# ...

def index(self, substr):
    """
    Returns the index of the first occurrence of substr in
    the string.

    If substr does not occur within the string, raises a
    ValueError exception.
    """
    # ...
```

Pydoc

Points to note:

- Instead of returning -1 when the string does not occur, Python throws an exception.
- In Python, this is a typical idiom: exceptions are often thrown to indicate the absence of something.
- More on exceptions later.

Python docstrings

- The Pydoc tool makes use of Python *docstrings*.
- If the first expression within a module, class, function or method is a *string*, then that is used as the *docstring* for that module (or class or function or method).
- Unlike Javadoc, Pydoc does not have special markup for documenting parameters or return values – but more comprehensive documentation tools exist (the chief one being Sphinx) which do.

Documenting a Python function

- So you can document a *function* by making the first expression in the function a string:

```
# ...

def myFunction(myArg):
  """
  This function frobnicates the argument "myArg"
  """
  # ...
```

Documenting a Python class

- And you can document a *class* by making the first expression in the class a string:

```
# ...
class MyClass:

  "The MyClass class provides a frobnication service"

  # ...
```

- Similarly for Python modules.

APIs

- The specification for all the externally accessible classes, methods and so on in a module make up what is called the *API*[3] of the module – the "Application Programming Interface".

[3] See further "Who invented the API?", https://nordicapis.com/who-invented-the-api/

APIs

- The specification for all the externally accessible classes, methods and so on in a module make up what is called the *API*[3] of the module – the "Application Programming Interface".
- The name derives from the idea that if we write a re-usable component of some sort (like a library), then other developers will want to use this in their application programming, and we should document the public *interface* to that component.

[3] See further "Who invented the API?", https://nordicapis.com/who-invented-the-api/

APIs

- The specification for all the externally accessible classes, methods and so on in a module make up what is called the *API*[3] of the module – the "Application Programming Interface".
- The name derives from the idea that if we write a re-usable component of some sort (like a library), then other developers will want to use this in their application programming, and we should document the public *interface* to that component.
- (Actually, the other developers might not be writing an *application* per se – they might be writing another library – but the name has stuck.)

[3] See further "Who invented the API?", https://nordicapis.com/who-invented-the-api/

APIs as contracts

- We can think of the API for a function (or other procedural unit) as constituting a *contract* between the developer of the function, and the client code using it.[4]
- In effect, the documentation says "If you, the client code, pass me arguments which meet the following criteria, I promise to do the following thing: . . . "

[4] See further "Design by contract", https://en.wikipedia.org/wiki/Design_by_contract; Meyer, Bertrand. "Applying 'design by contract'." Computer 25.10 (1992): 40-51.

APIs, cont'd

- The "following thing" – the behaviour of the function – will usually be to return some sort of value, or to cause some sort of "side effect".
- (A *side effect* is anything the function does to alter the current or subsequent behaviour of the system or its interaction with external systems, other than returning a value.
  For example, writing a file to disk, or sending an email, or changing the value of a global variable.)

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

  `void println(String x)`
  Prints a String and then terminate the line.

API example

- If you have used Java, you most likely at some point will have written something like
  `System.out.println("some string ...")`
- What does the `println` method promise to do?
- The Javadoc says:

  `void println(String x)`
  Prints a String and then terminate the line.

- Does `println()` return a value? If not, then what does it promise to do?

APIs – specification vs implementation

The API documentation does not normally say *how* the function is to be implemented – just what its return value and effects are.

This means that if the library developer decides to reimplement the function in another way (for instance, to improve efficiency), they can, without changing the API.

Specification vs implementation example

In fact, you can have multiple implementations of the same API by different developers.

Example:

- Oracle corporation provides an implementation of the Java standard libraries (as well as of the Java compiler, `javac`, and the Java Virtual Machine or JVM).
- But there are other implementations – for instance, OpenJDK, an open-source version of the standard libraries.
- These adhere to exactly the same specifications as the Oracle versions.
- (In fact, since Java version 7, the OpenJDK version has been the reference implementation

Specification vs implementation – other examples

- The POSIX standard specifies an API for Unix-like systems, and has been implemented multiple times in different ways by different operating systems. (In fact, even Windows, at various times, has met the POSIX standards.)

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.
- But there's nothing in the *specification* to stop us from implementing it in other ways . . .

Specification vs implementation in Java

- Q. In Java, does the API tell us *how* `String.indexOf(ch)` is implemented? How would you implement it?
- A. It does not. The *plausible* way to do it is to test each possible index from 0 to (length-of-string - 1), see if matches the character we're looking for, and if it does, return the index we're currently at.
- But there's nothing in the *specification* to stop us from implementing it in other ways ...
- e.g. Generate a random number from 0 to (length-of-string - 1), call it k. Check and see if we've hit all positions from 0 to $k-1$ yet; if we have, and `inputString.charAt(k)` equals the character we're after, return the current index.[5]

[5] See also Bogosort, https://en.wikipedia.org/wiki/Bogosort

Specification vs implementation – "illities"

Sometimes specifications for units will describe not just what the unit *returns* or *does*, but *how it does it*.

For instance, if we implement `String.indexOf(ch)` in the "generate a random number" way we described, it would be *extremely* slow.

Specification vs implementation – "illities"

The specification of `String.indexOf(ch)` could rule out "silly" implementations like this, by saying something like "the `indexOf` method shall provide guaranteed $O(n)$ time cost, where n is the length of the string".

(If you have not done a data structures and algorithms course, don't worry too much about what "$O(n)$" means – it roughly means that the time to execute `indexOf` will increase in proportion to the length of the string.)

Specification vs implementation – "illities"

If you look at the documentation for Java's `TreeMap` class, in fact, you will see that the implementation provided by Oracle promises to provide guarantees about how long particular methods will take to run:

> *This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's Introduction to Algorithms.*

APIs, summary

So:

- The API describes the *expected behaviour* of a module (or larger system).
- The code constitutes a particular *implementation* of that API.

APIs, cont'd

What should go in the API documentation?

- The *preconditions* – any conditions which should be satisfied by the parameters or the system state when the function is called.
- The *postconditions* – the return value of the function, and any *changes* the function makes to the system state (the "side effects" discussed earlier)

We often also will document what will happen if the preconditions *aren't* satisfied: in many languages, this will typically be an exception being thrown.

APIs, cont'd

Once we know the preconditions and postconditions for a function, we can write tests for it.

(They needn't be spelled out formally or mathematically – but it is best if they are clear, consistent and unambiguous.)

Unit tests

Unit tests

Unit tests should focus on one tiny bit of functionality, and attempt to find any deviations from expected behaviour.

Desirable properties of unit tests

Ideally, unit tests should be -

- quick to run. We want developers to run tests whenever changes are made to the code, or at least when they are committed to version control.
- independent of other tests. Tests should not rely on other, particular tests having been run before them.
- run frequently. We want to identify faults as early as possible!
  - Most version control systems make it possible to perform particular tasks whenever code is committed, using "hooks"
  - It's therefore possible to run tests every time code is committed. (But if tests aren't quick to run, developers may avoid committing code regularly.)

JUnit and xUnit

- One of the best-known unit-testing frameworks is JUnit.

JUnit and xUnit

- One of the best-known unit-testing frameworks is JUnit.
- JUnit derives from a similar framework developed for Smalltalk by Kent Beck, named SUnit.

JUnit and xUnit

- One of the best-known unit-testing frameworks is JUnit.
- JUnit derives from a similar framework developed for Smalltalk by Kent Beck, named SUnit.
- The same general framework has been implemented in a huge array of other languages:

JUnit and xUnit

- One of the best-known unit-testing frameworks is JUnit.
- JUnit derives from a similar framework developed for Smalltalk by Kent Beck, named SUnit.
- The same general framework has been implemented in a huge array of other languages:
  - C# (e.g. NUnit)
  - Python (e.g. unittest, sometimes called "PyUnit")
  - Go (go2xunit)
  - Haskell (e.g. HUnit)
  - Lua (e.g. LuaUnit)
- Collectively, these frameworks are sometimes referred to as "xUnit"

Unit testing – Java example

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
// ...

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Unit testing – Java example

- In Java, methods which are intended to be run as tests are labelled with the *annotation* `org.junit.Test`.
- The test framework can then be used to run a test.
  e.g.

```
$ java -cp .:junit-4.01.jar org.junit.runner.JUnitCore
CalculatorTest
```

Unit testing – Python example

- Using `unittest`, classes containing tests inherit from `unittest.TestCase`, and methods constituting tests begin with the letters "test":

```
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(3), 4)
```

Unit-testing terminology

- test case – the basic unit of testing, which checks the behaviour of code in response to a particular set of inputs. It consists of one particular set of input data, and the expected output (behaviour).

Unit-testing terminology

- test case – the basic unit of testing, which checks the behaviour of code in response to a particular set of inputs. It consists of one particular set of input data, and the expected output (behaviour).
- test suite – a collection of test cases (or other test suites)

Unit-testing terminology

- test case – the basic unit of testing, which checks the behaviour of code in response to a particular set of inputs. It consists of one particular set of input data, and the expected output (behaviour).
- test suite – a collection of test cases (or other test suites)
- test runner – a software tool which manages the execution of tests, and reports their outcome

Unit-testing terminology

- test case – the basic unit of testing, which checks the behaviour of code in response to a particular set of inputs. It consists of one particular set of input data, and the expected output (behaviour).
- test suite – a collection of test cases (or other test suites)
- test runner – a software tool which manages the execution of tests, and reports their outcome
- test fixture – the preparation needed to perform one or more tests

Test fixtures

The idea of a "fixture" comes from testing of hardware – a "fixture" is everything that holds the piece of hardware in place, and provides you with known environment and conditions it can be tested in.

Test fixtures

- For software, we likewise may need to get the environment and conditions into a known state for testing.
- Things we might need to do:
  - Prepare input data (it may not be just simple numbers or strings – it could be an MS Word document, say, or some complex data structure)
  - Create fake or mock objects (used to deal with dependencies – more on these later)
  - Load a database with a specific, known set of data
  - Create files with known contents
  - . . . etc.

Framework features

Most unit-testing frameworks provide the ability to -

- collect related tests together (e.g. into suites)
- identify and run all unit tests (or suites) in a module, or the whole system
- produce output in different forms (e.g. human-readable text, XML, HTML)

Expected behaviour

- What sort of behaviours might we expect from code under test?

Expected behaviour

- What sort of behaviours might we expect from code under test?
  - return of a value

Expected behaviour

- What sort of behaviours might we expect from code under test?
  - return of a value
  - alteration of state

Expected behaviour

- What sort of behaviours might we expect from code under test?
  - return of a value
  - alteration of state
  - throwing of an exception

Expected behaviour

- What sort of behaviours might we expect from code under test?
  - return of a value
  - alteration of state
  - throwing of an exception
- Basically, the same things that we would document as *postconditions*.

Expected behaviour

- What sort of behaviours might we expect from code under test?
  - return of a value
  - alteration of state
  - throwing of an exception
- Basically, the same things that we would document as *postconditions*.
- Unit testing frameworks will typically provide ways of detecting all of these, and comparing them with expected results.

Indicating what the tests are

We need to indicate to the test runner that something is intended to be a test. Typical ways are:

- annotations (example – JUnit 4.x)
- inheritance (example – Python unittest)
- naming conventions (example – Python unittest, cppunit)

References

- Bruegge and Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java* (Pearson, 2010)
- Martin Fowler, "Mocks Aren't Stubs" (https://martinfowler.com/articles/mocksArentStubs.html)
- Gerard Meszaros, *xUnit Test Patterns: Refactoring Test Code* (Addison-Wesley Professional, 2007)
- Claessen and Hughes, "QuickCheck: a lightweight tool for random testing of Haskell programs." *ACM Sigplan Notices* 46.4 (2011): 53-64.
- Kristopher Sandoval, "Who Invented the API?", Sept 20 2018 (https://nordicapis.com/who-invented-the-api/).