CompSci 230
Software Design and Construction

Software Quality 2015S1

Black box testing

Lecture plan

Week 1: *No class - Anzac Day*
What is software quality?
Some key developer practices (version control, testing).

Week 2: Black box testing.
White-box testing.
Myers' testing principles.

Week 3: Traditional approach to testing (Waterfall).
Agile approach to testing (XP).
Famous failures.

Myers Ch. 2, pp. 8-10

JUnit Tutorial v2.3 by Lars Vogel, 2012-12-06, sections 1-3.3

Learning Goals for Today

- Develop a "working understanding" of Myers' theory of the economics of software testing.
  - Can we simply 'test everything,' just to be sure?

- Develop a test suite for some code without looking at it. (Black-box testing.)
  - What are your initial questions? (Have you written some already?)
  - Should I try to "write it myself" (to discover some likely bugs)?
  - Should I carefully test interfaces, exceptions, or error returns, or should I concentrate on confirming the correctness of functional "internal" behaviour?

Triangle-classification program

- Last session, you were challenged to think about how to test a small program.

- Program description:
  - "The program reads three integer values from an input dialog.
  - "The three values represent the lengths of sides of a triangle.
  - "The program displays a message that states whether the triangle is scalene, isosceles, or equilateral."

- Recall:
  - A scalene triangle is one where no two sides are equal.
  - An isosceles triangle has two equal sides.
  - An equilateral triangle has three sides of equal length.

Triangle-classification program

- My thought process (yours will be different!): this program has at least three outputs: "scalene", "isosceles", "equilateral"…
  - I know that a set of test cases doesn't have complete coverage if it doesn't include at least one case for each program output.
  - I can easily write three test cases: a valid input for a scalene triangle, a valid input for an isosceles triangle, and a valid input for an equilateral triangle.
- Be careful!
  - "Do you have a test case that represents a *valid* scalene triangle? (Note that test cases such as 1,2,3 and 2,5,10 do not warrant a "yes" answer because there does not exist a triangle with these dimensions.)"
  - (Hmmm, I wrote "1,2,3". I thought this was an interesting – because degenerate – case of a scalene triangle ;-)
  - Process question: how can I be sure my test cases are correct?

Myer's evaluation scheme

- 1,2,3: do you have valid test cases for the three types of triangle?
  - My result: no I didn't in all cases, but I *thought* I did. And my mistake revealed an ambiguity in the program specification!
- 4: "Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of three equal sides (such as, 3,3,4; 3,4,3; and 4,3,3)?"
  - My result: no.
  - Why does Myers think this is important? Hmmm… now I get it! (Do you?)
- 5: "Do you have a test case in which one side has a zero value?"
  - My result: no, I didn't test for a degenerate (zero-area) isosceles triangle. This is an important boundary case which I should have tested. Ooops!

Myers' Evaluation of a Set of Test Cases

- 6. Do you have a test case in which one side has a negative value?
  - Ouch, I really should have thought of that! I've been burned by that sort of latent bug before… !@#^&* unsigned ints in C…
- 7. Do you have a test case with three integers greater than zero such that the sum of two of the numbers is equal to the third?
  - Well, no, but I'm finally remembering the "triangle inequality":
    - For any triangle, the sum of the length of any two sides is greater than or equal to the length of the third side.
    - For triangles drawn on a Euclidean plane, the inequality is strict: "For any non-degenerate triangle, the sum of the length of any two sides is greater than the length of the third side."
  - I finally know enough about the problem to write a good test suite!
    - If you don't discover the "boundary cases" you probably won't test them…

Continuing with Myers' Evaluation

- 7. ..."(That is, if the program said that 1,2,3 represents a scalene triangle, it would contain a bug)."
  - Oh… now you're telling me! I would have called it a "degenerate scalene triangle" – it's still scalene, but has zero area. But if you don't want to call it a triangle, that's fine by me: you write the spec, and I test it. I'll adjust my test cases now.
- 8. Do you have at least three test cases in category 7 (i.e. a degenerate triangle with zero area) such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides (for example, 1,2,3; 1,3,2; and 3,1,2)?
  - No… but I see what you're getting at. An input that is "strange" in one way (e.g. degenerate) may provoke strange behaviour in the program, so I should test such cases carefully (provoking all possible program outputs; different input permutations; …)

Writing Test Cases is an Iterative Process!

- The initial set of test cases will, usually, trigger some questions about what the program is "designed to do" and what is "unintended".
  - Most program-specifiers don't think about what is "unintended".
  - Most program-specifiers assume that "everybody" will know what they mean by a "simple" word such as "triangle" or "input".
  - Test cases usually reveal ambiguities and gaps in the specification!
- But: you'll never get a job done if you keep asking questions.
  - Usually you have to "steam ahead". If you're methodical, you'll write down your unanswered questions, and prioritise them. (Should a tester be methodical?)
  - Which questions are important enough that they really *must* be answered *before* I deliver my first-draft test set? (My experience: none!!!!)

Myers Evaluation: it just keeps going…

- 9. Do you have a test case with three integers greater than zero such that the sum of two of the numbers is less than the third (such as 1,2,4 or 12,15,30)?
  - Yes, I wrote this case. But your spec didn't say how the program is supposed to behave when it gets an input that isn't a triangle. I assumed the program should terminate normally, and output nothing. (Is this what you want it to do? I can't test for the "correct" production of an unspecified output.)
- 10. Do you have at least three test cases in category 9 such that you have tried all three permutations (for example, 1,2,4; 1,4,2; and 4,1,2)?
  - No, I didn't think about permuting that case. But it's very important, come to think of it! If the program doesn't check all three triangle inequalities (A+B<C, A+C<B, and B+C<A) then it'll accept some non-triangle inputs.

… and going… who would have guessed?

- 11. Do you have a test case in which all sides are zero (0,0,0)?
  - No. (Why is this important, I wonder? Ah… a "boundary" with the negative-length case!)
- 12. Do you have at least one test case specifying non-integer values (such as 2.5,3.5,5.5)?
  - Well, I did a little checking of the keyboard input, but my test was for integers that would overflow a 4-byte signed int: 10000000000, 20000000000, 40000000000. I also gave some thought to checking for semicolons, commas, and spaces as delimiters but I reckoned that this was my first-draft test set for a program with an unspecified input format. I'd prefer to see the first set of testing results before finalising my tests.
  - Is the program written in C, Java, Fortran, PHP? Different programming languages handle integer input quite differently, leading to different confusions over "unintended behaviour".

… more evaluative questions from Myers!

- 13. "Do you have at least one test case specifying the wrong number of values (two rather than three integers, for example)?
  - Well, I certainly thought about writing one, then decided that was a test case for the "input dialog module". Now that I know I'm supposed to test a program that is handling its own keyboard input, I'll add more cases. I have been making a lot of incorrect assumptions about the input format you designed your program to handle. Do you want to tell me about your format now?
- 14. "For each test case did you specify the expected output from the program in addition to the input values."
  - Yes, But I incorrectly guessed what was "expected" for degenerate triangles. And I still don't know what the program was designed to output in cases where the input is not a triangle. So some of my output specifications are incorrect.

Myer's Summative Metric

- Myers gives one point for an affirmative answer to each question. Max 14. Min 0. Average (for "highly qualified professional programmers") 7.8.
  - My score: 4. (Do you really want to listen to my lectures on testing?)
- "The point of the exercise is to illustrate that the testing of even a trivial program such as this is not an easy task."
  - Was his exercise successful for you? For others? (You could test this!)
- "… consider the difficulty of testing a 100,000-statement air traffic control system …"
  - Ouch. The text shows its age! This passage was probably written for the first edition and hasn't been updated…
  - This introductory chapter raised some very interesting questions and provided plausible definitions; but is Myers' advice still relevant?

Some Recent Software Systems

- "An entirely new architecture is needed, which may mean moving away from a central computer to a more distributed client-server system. That would enable the FAA to upgrade the host in bite-sized chunks, rather than recoding all at once the 1-1/2 million lines of code that got us into trouble in the first place."

  [In Search of the Future of Air Traffic Control", *IEEE Spectrum*, August 1997]

- "a typical cellphone now contains 2 million lines of code; by 2010 it will likely have 10 times as many… [and] cars will each have 100 million lines of code."

  ["Why Software Fails", *IEEE Spectrum*, September 2005]

- "… about 8.6 million lines of Android's 11 million are open-source."

  ["Google carves an Android path through open-source world", CNET News, 22 May 2008]

- Can anyone afford to write ten million test cases for a single program? Can we rely on programs that aren't fully tested?

Testing observations

- Can we make some sense of what we discovered when trying to test the triangle-classification program above?

- Some observations. We :
  - had to check that each input was valid (e.g. not negative or zero)
  - had to check that the combination of inputs was valid (sum of length of 2 sides > length of 3rd)
  - did not know what to do for an invalid set of inputs (1,2,4)
  - did not understand the user's viewpoint (degenerate case)
  - we had to check all permutations of the inputs
  - had to check the program correctly categorised the inputs (functioned correctly)

Economics of Software Testing

- "In an ideal world, we would want to test every possible permutation of a program.
  - In most cases, however, this simply is not possible.
  - Even a seemingly simple program can have hundreds or thousands of possible input and output combinations."
- "Creating test cases for all of these possibilities is impractical"
  - Why? Can't a computer program help me generate tests for 10k possibilities? But I can see a feasibility problem if there are more than a billion possibilities...
- "Complete testing of a complex application would take too long and require too many resources to be economically feasible."
  - I agree – completely testing a 32-bit multiplier is infeasible, and that's not even a complex application!
  - Hmmm… is there an objective way to determine the "dollar value" of a software test? Is testing an art, a craft, or an engineering discipline?

More on the Economics of Testing

- If you accept that the primary purpose of testing is defect-identification (rather than for sales-support ;-), then… what is a cost-effective way to test?
- Myers suggests that the appropriate first step (by the tester or their manager) is to make a strategic decision. Two of the most common strategies:
  - Black-box testing
  - White-box testing

Black-box Testing

- In black-box, data-driven, or input/output-driven testing you should
  - "… view the program as a black box.
  - "Your goal is to be completely unconcerned about the internal behavior and structure of the program.
  - "Instead, concentrate on finding circumstances in which the program does not behave according to its specifications. …
  - "test data are derived solely from the specifications (i.e. without taking advantage of knowledge of the internal structure of the program)."

- (Is this the first testing strategy you thought of, while reading this Chapter? Can you think of any strengths or weaknesses?)

Evaluating a Black-Box Test Set

- To have a chance of finding all errors, you could use *exhaustive input testing*: making use of every possible input condition as a test case."
  - This is a "gold standard" for defect-detection – but infeasible, in nearly all cases of practical interest.
    - Could you exhaustively test the triangle-classification program of Chapter 1?
  - Exhaustive test is desirable: if you don't test all possible inputs, then the program may exhibit a bug on any of the inputs you don't test.
  - True or false: if you test all possible inputs, and the program passes all of your tests, then you have demonstrated that the program has no bugs.
    - A program may write to disk, behaving differently on subsequent runs.
    - The program may be multi-threaded, with an occasional race leading to an incorrect answer, or an occasional deadlock.
    - A test case may have an error.

Maximising "Yield on Investment"

- The "objective should be to maximise the yield on the testing investment by maximising the number of errors found by a finite number of test cases."
- "Doing so will involve, among other things,
  - being able to peer inside the program and
  - making certain reasonable, but not airtight, assumptions about the program
  - (for example, if the triangle program detects 2,2,2 as an equilateral triangle, it seems reasonable that it will do the same for 3,3,3).
- This seems to contradict an earlier statement:
  - "Your goal is to be completely unconcerned about the internal behavior and structure of the program."

Is Myers Advocating "Grey-box testing"??

- Myers seems to say you must be able to peek at (or rather "peer inside") a program, in order to write a good set of black-box tests?
  - I think Myer does not mean this!
- A tester shouldn't peek;
  - but they can (and I think should!) guess at the code a programmer is likely to write, then
    - write some cases to "catch the errors" which might occur if the programmer wrote a buggy version of this code.
  - For example, when a programmer is writing a conditional test for isosceles triangles,
    - they might test whether the first two integers are equal, and whether the last two integers are equal, but
    - they might "forget" to test whether the first and third integers are equal.

Black box strategies

- Equivalence partitioning:
  - "A test case should invoke as many different input considerations as possible to minimize the total number of test cases necessary."
  - "You should try to partition the input domain … into a finite number of equivalence classes such that you can reasonably assume … that a test of a representative value of each class is equivalent to a test of any other value."

Myers Ch. 4, pp. 50

Black box strategies

- Example:
  - Valid input is integer in range 3 – 10.
  - 3 tests
    - out of range at lower end (say, 1)
    - in range (6)
    - out of range at higher end (15)

- Example:
  - Valid input is character string, first character must be a letter.
  - 2 tests
    - first character is a letter
    - first character is not a letter

Black box strategies

- Boundary value analysis:
  - "Rather than selecting any element in an equivalence class as being representative, … elements (should) be selected such that each edge of the equivalence class is the subject of a test."
  - "Rather than just focussing attention on the input conditions (input space),test cases are also derived by considering the *result space* (output equivalence classes)."

Myers Ch. 4, pp. 55

Black box strategies

- Example:
  - Valid input is integer in range 3 – 10.
    - out of range at lower end (2)
    - in range (two tests - 3, 10)
    - out of range at higher end (11)

- Example:
  - Output is ordered list of items.
    - focus on first and last elements in the list

Some scenarios

- Based on the earlier observations with the triangle-classification program, can you think about how you might go about designing black-box tests for a program with the following characteristics:
  - a string input
  - an input that is a list of items
  - the program adds/removes items to/from a list

A Very Brief Introduction to JUnit

- JUnit is a set of "software tools" for unit testing.
  - Kent Beck adapted it from his earlier SUnit [K. Beck, "Simple Smalltalk Testing: with Patterns", chapter 30 of *Guide to Better Smalltalk*, 1998].
- The syntax and semantics of JUnit are variable, depending on the release version.
  - Old tests must be ported and re-validated, if you're using a new version of Java or a new version of JUnit.
- A distraction: "Write Once, Run Anywhere" (WORA) does not imply "Write Once, Run At Any Time in the Future"!
  - A cynical joke: "Write Once, Debug Everywhere".
    - (Microsoft's JVM wasn't the same as Sun's. Apple's JVM isn't the same as Oracle's… See http://www.uberpulse.com/us/2008/05/java_write_once_debug_everywhere.php.)
  - It is possible to write very portable Java, and very portable JUnit tests.
    - You should use only basic features and standard libraries!

A Test Fixture in JUnit

- By convention, if you're testing Xxx your extension should be called testXxx (or XxxTest).

```
import org.junit.*;
public class testYourClass { \\ tests for YourClass
  @Before public void setUp(){
    \\ allocate some objects for use during test
  }
  \\ put your test cases here
  @Test public void testCheckPrime() {
    assertFalse(check4prime.primeCheck(0)); \\ Test Case #1
    assertTrue(check4prime.primeCheck(3));  \\ Test Case #2
  }
  @Test(expected=IllegalArgumentException.class,timeout=100)
        public void testCheckPrimeRed() {
    assertTrue(check4prime.inputValidator("1,000")); \\ Test Case #3
  }

  @After public void cleanUp() { \\ de-allocate your test setup
  }
}
```