Software Testing IOOP Lecture 3
Testing in the Large

Justin Pearson

September 30, 2011

Lecture Plan

- Some more testing techniques
- Testing and the Software development process
- Test plans and some ideas about how testing might be organised at a large company.

Pop Quiz

- How much testing is enough?

Pop Quiz

- How much testing is enough?
- In theory you can never do too enough testing. Remember testing shows the presence of bugs not the absence.

Pop Quiz

- How much testing is enough?
- In theory you can never do too enough testing. Remember testing shows the presence of bugs not the absence.
- In practise, to decide if you have done enough testing is an exercise in *risk assessment*.

Risk Assessment

- Have I covered all the code?
- Have I tested all the requirements?
  - The customer wants feature X
  - How do I test for feature X?
- In places where there is complicated logic in the code have I tested all possible combinations.
  - If your code is too complicated to test then, simplify the code so you can test it.

Black Box vs White Box

- Black box testing :- Look at the functionality of the software rather than its internals.
- White box testing :- Look inside the software.

Code Coverage

- Code coverage is actually quite a complicated problem.
- It is not enough for every line of code to be executed. You want every execution path to be covered.
- There is in general too many execution paths.
- Some approximations are used.
- Practically you need test cases that test every function/method. You need test cases that test every branch, loop.
- There is tool support.

Logic Coverage

- Code coverage:

```
if condition:
    ......
else:
    .....
```

- Then you have to find a test case that makes the expression evaluate to true and a test case that evaluates to false.
- But if the logical expression is more complicated then:

$$(\phi \wedge \psi) \vee \chi$$

- Then to be complete you have to evaluate all the 2^n possible conditions (where n is the number of clauses).
- But this can lead to too many test cases.

Logic Coverage

- There are many approximations to testing all cases:
- Find a test case where each clause evaluates to true and a test case where each test case evaluates to false.
- Find assignments where flipping the truth value of a clause has an effect.
  - For example in:

    $$(\text{True} \vee \phi)$$

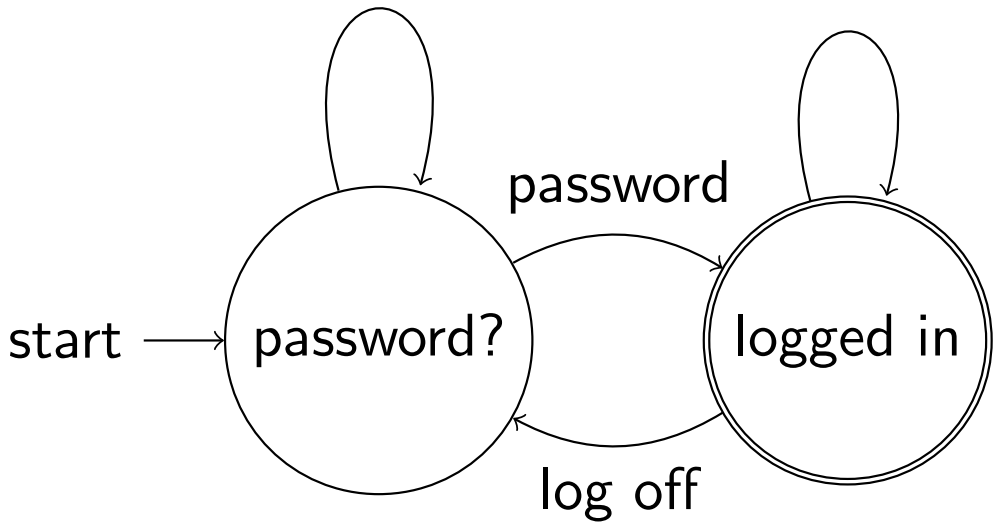
    Flipping ϕ to true or false has no effect on the overall truth value.
  - While flipping ϕ in

    $$(\text{False} \vee \phi)$$

    has an effect.

State Machines Coverage

Often you model aspects of software as a finite sate machine:



Find test cases that go through every path in the automaton.

Input Values

- Check boundary values.
- Extreme values: empty strings, lists, `maxint` etc.

Input Space Partitioning

- Think about the problem domain, model the inputs.
- Partition the inputs into sets of similar inputs.
- Pick an item from each partition.

For example when testing a sorting function you might pick as inputs:

- Empty list
- A list of length 1
- A list of average length (what ever that means for you application).
- A very large list.

It does not really matter what the values are. You do not have to test for every list.

Syntax Based Testing

Mutation based testing.

- Mutate the program to something slightly different.
- Find a test that distinguishes the mutant code from the original code.

Mutation Testing

```
if a and b:
    c = 1
else:
    c = 0
```

Mutate the code to:

```
if a or b:
    c = 1
else:
    c = 0
```

Find a test case (for example *a* is set to True and *b* is set to False) that causes the mutants to behave differently. This is not as crazy as it sounds. It does uncover interesting errors.

Software Engineering

- There are many software processes out there. But in essence producing software is a path between requirements and code.
- In different parts of the software engineering process different testing regimes are required:
  - Acceptance Testing :- Does my software meet its requirements?
  - System Testing :- asses software with respect to architectural design.
  - Integration Testing :- asses how various parts of the software fits together.
  - Module Testing :- Test each individual module's interfaces.
  - Unit Testing.
- Regression Testing :- I have made a change does my software still work?

Requirements gathering

- Even during requirements analysis you can think about testing. Work out how you phrase your requirements to be testable.
- For example
  - Requirements :- “the graphics should be awesome” becomes
  - The frame rate should always be more than 30fps, the resolution should be ... etc.

Integration Testing

- You can not really say that much in general it depends on the design of your system.
- You often have to simulate parts of the system with stubs and drivers.
- You might have to roll your own test scripts.

Regression Testing – Managing test suites

- As a project grows so do your tests.
- You do not want to wait for ever for your tests to run.
- For each test you have to decide if it still needs to be there or it is covered by other tests.
- There is no easy way.

Testing in the large

- A part of life in a large company is documentation.
- Test plan:

  > *A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning*

- There is a whole IEEE standard (very boring).
- Take home message, document how you should test, what you should test and a risk analysis (admit that you can not test for every thing, say what you think that you might be missing).
- Many companies do not have a coherent testing strategy.

Documenting test results

- This can be partly automated.
- What do you do when a test fails?

Documenting test results

- This can be partly automated.
- What do you do when a test fails?
- Incorrect answer: fix the bug. (This is only true when you are doing TDD or small scale development)

Documenting test results

- This can be partly automated.
- What do you do when a test fails?
- Incorrect answer: fix the bug. (This is only true when you are doing TDD or small scale development)
- Correct answer:
  - Decide if it really is an error.
  - Decide if it worth fixing the bug, can it wait? Do you have more important stuff to do?
  - If you are going to fix it then document and recommend the fix to the programmers.

Test Engineers

- Often the tester is a separate person from the programmer.
- Gives a fresh perspective. You think of test cases and the correct behaviour of the code without thinking about the code.
- Testing should not be an excuse for conflict. You need to be constructive. Your job is to improve the quality of the project.

Testing

- Learning to test is a life long process. Finding good test cases is often hard and requires experience.
- Start testing as early as possible. Make it part of your build process and make sure it is automated.
- Design your software so it is testable.