CITS5501 Software Testing and Quality Assurance
Semester 1, 2020
Notes from workshop 5 – Logic-based testing

These are my notes from today's workshop.

Full model solutions will be posted at the end of the week.

- General points about the unit.
  - Especially for the first half of the unit (covering unit testing and the types of model-based testing – ISP, graph-based, logic-based, and syntax-based) – when answering questions, concise is usually good. Often questions about these topics can be answered pretty briefly.
  - For any software engineering unit, we're interested in things that make it easier or harder to deliver solutions that meet requirements, are on-time, and on-budget.

    The ultimate explanation for *why* we do things one way rather than another is nearly always because it makes it easier to deliver such solutions.
  - e.g. Why is it a good thing for tests to be independent? Answer: Just because, if I have 2 tests A and B, and B only runs correctly when A has run, and both tests fail – then I now don't know exactly why they failed and have to spend more time tracking down the cause. And if I have hundreds of tests that depend on each other in various ways, tracking down the reason for failures becomes very difficult. So the ultimate reason for making tests independent is: it makes it quicker to isolate problems with failed tests, and thus help us deliver software on-time that passes our tests.
    If you look up either of the textbooks (Amman and Offutt, or Pressman), they'll say something along these lines (as will any textbook on testing).
- Where to look besides the lecture slides
  - If something in the slides isn't clear, then for the first half of the unit, the best place to look for more context or more information is the Amman and Offutt textbook.

    (However: that textbook doesn't have very good coverage of failures/faults/erroneous states, nor of preconditions/postconditions/invariants – so on the lecture schedule for the first couple of lectures, I've got readings for some textbooks that do. The readings are all available as PDFs from the Unit Readings site.)

– And obviously if that doesn't help, feel free to post on the forum and ask me questions. If people aren't understanding something, I won't know unless you tell me.

Workshop questions

Question 1(a)

What are the clauses?

They are:

- $f \leq g$
- $x > 0$
- M
- $(e < d + c)$

A clause is anything that evaluates to a boolean, and *doesn't* contain any logical connectives (e.g. "and", "or", "implies", "if and only if").

A predicate consists of one or more clauses joined by logical connectives (and, often, parentheses). So the *simplest* predicate is actually just a clause on its own.

Making clauses active

Let's look at how we make the clause containing just "M" active.

Our predicate is:

$((f \leq g) \wedge (x > 0)) \vee (M \wedge (e < d + c))$

One way to tackle this is to work "from the inside out":

- Look at *just* the bit of the predicate $M \wedge (e < d + c)$. How do we make M active? We need $e < d + c$ to be true.
- What about $M \wedge (e < d + c)$ – how do we ensure that makes a difference to the result?
  Well, it's joined by an "or" to $((f \leq g) \wedge (x > 0))$, so we need to make sure that $((f \leq g) \wedge (x > 0))$ is *false*.
- How do we make it false? It'll be false when either $f \leq g$ is false or $x > 0$ is false.

Therefore – to make M active, we need:

- $e < d + c$ to be true
- one or both of $f \leq g$ and $x > 0$ to be false

We aren't actually told what e, d, c, f, g and x are, so let's make (what to me seems like) the simplest assumption and assume they are `int`s.

One set of inputs that makes M active, then, would be:

- $e = 1, d = 2, c = 2$, to make $e < d + c$ true
- $f = 2, g = 1, x = -1$, to make $(f \leq g) \wedge (x > 0)$ false

It's usually a good idea to show your working – then, if you made a mistake and don't get quite the right answer, you can still get marks for how you've worked out the solution.

(Also, in this case, adding the working out/explanation makes it easier for a marker to see what you're doing and why you chose the values you did.)

Question 2

For this question, we're going to have to assume we have *some* way of finding out, for a particular product we sell:

- is it a mouse?
- what does it sell for?
- is it wireless?
- how many do we have in stock?

If we assume that the eventual system delivered is in some object-oriented language, then we might further assume that we've got some sort of class `Product`, with methods like:

- `int retailPrice()`
- `String productType()`
- `bool isWireless()`
- `int currentStockLevel()`

(In reality, the system might be implemented in some non-OO language, and our predicate might instead eventually be implemented in some language like the SQL database query language – but let's make assumptions that make life simpler for ourselves. We may as well also assume prices are in dollars. Not all that plausible, but ok for a question like this.)

Then that means that a predicate that satisfies the requirement might be something like – assuming we have some object `o` of type `Product`:

$$o.productType() = \text{"mouse"} \wedge \\ (((o.retailPrice > 100) \wedge (o.isWireless() \vee o.currentStockLevel > 20)) \\ \vee (\neg o.isWireless() \wedge o.retailPrice > 50)$$