Verification and validation

Most of the material we need is in Using UML:

Book reading:
Chapter 19

We reviewed verification ("are we building the product right?") and validation ("are we building the right product?").

We discussed sanity checking of UML models, which is an easy but important part of verification. For example, because the set of classes in a system must be well defined, class names that appear anywhere should also appear in the class model. In a state diagram for a class, messages appearing as events (before a slash) should correspond to operations available on the class (that is, messages that are understood by objects of the class). Messages appearing as actions should correspond to messages understood by the object to which they are sent, which in turn should belong to a class which is associated by a navigable association to the class whose state diagram we are studying. An example may make this clearer.

If in a state diagram for class A, a state transition from state $s1$ to state $s2$ carries the label $a()/o.b()$, this indicates that if an object of class A is in state $s1$ and it receives the message $a()$, its reaction will be to send the message $b()$ to object o. It is not completely settled what the formally legal options for o are. Morally, it should be a name for an object which this object of class A has access to. For example, if the class diagram shows A associated by a navigable 1-1 association to a class O, you might label the O end of the association with (the role name) o; this would indicate that o is the name which will be used as the name by which an object of class A knows the corresponding object of class O. You should probably either make sure that there are such role names on the class diagram, or make it clear using notes which object is supposed to receive the message.

Testing

Slides briefly reviewing material on testing that most of you covered in PM, plus a discussion of what's special about object oriented programs.

Testing

You should remember (from PM) the process aspects of testing.

Ways of testing: black box (specification-based) and white box (structural).

Different testing purposes:

- Module (or unit) testing
- Integration testing
- System testing
- Acceptance testing
- Stress testing
- Performance testing

and maybe some techniques, like equivalence partitioning?

Slide 41

Why test?

Testing has three main purposes:

- to help you find the bugs
- to convince the customer that there are no bugs
- to provide information that will help with system evolution.

The first one is the hard one!

A successful test is one that finds a bug.

Only once you can't find bugs does it make sense to try to convince the customer that there aren't any. (Honest!)

Slide 42

Test what?

From the customer's point of view:

- Testing a system's capabilities is more important than testing its components.
- Testing old capabilities is more important than testing new capabilities.
- Testing typical situations is more important than testing boundary value cases.

Petschenik, H.N., *Practical priorities in system testing*, IEEE Computer, 2(5) 18-23.

Often overlooked: customer documentation testing.

Slide 43

How to test

Need to test old functionality: regression test. Therefore however tedious it is, tests must be:

- repeatable
- documented (both the tests and the results)
- precise
- done on configuration controlled software

Ideally, test spec should be written at the same time as the requirements spec: this helps to ensure that the requirements are highly testable. It may seem backwards to consider testability when writing requirements but it's now standard.

E.g. may write requirements in numbered sentences and keep a tally of which test(s) tests which requirement(s). Use cases may help structure tests.

Slide 44

Why is testing so often done badly?

It's *very very* **boring**, especially the documention of the whole process. Automation may help.

It's very expensive, in time and hence money. (May exceed 50% of project time: 30% is about normal.) The alternative may be worse, but the cost may not show up until later...

It's often planned mostly for the end of the project, and gets squeezed by deadline pressure.

Customers may pressure developers to deliver on time and untested rather than late and tested. (What should you do?)

Slide 45

Special problems of OO

Two main extra problems at the unit testing level:

1. The unit is the class, which is harder to test than a function. It has state, on which its behaviour may depend – so can't test methods as though they were independent.

   Moral: try to avoid designing classes with complex statecharts.

2. Polymorphism. When an object of class A interacts with an object of class B, it may in fact be interacting with an object of any subclass of B – which may not even have been invented when class A is tested!

   (This problem is to my mind the most convincing justification for work on formal methods...)

   Moral: remember this problem, and keep inheritance for when its advantages outweigh the disadvantages.

Of course, because the state dependency is in objects, we hope to save effort at integration and system testing, and to win overall.

Slide 46

But that's not the whole story

Problem: it's very hard to describe complex tests properly. How to handle bugs that only show up if you do a complex sequence of things?

- write well-modularized and module-tested software so that this is less likely to happen in the first place!
- have someone testing software with the aim of breaking it, however sneakishly.

(E.g. particular problem of testing GUIs: if you can come up with a really good solution to this one, patent it!)

That is, precise short documented tests are essential, but not sufficient.

The Black Team.

Slide 47

Formal technical reviews

What is a formal technical review?

A review or inspection is a meeting which more or less systematically attempts to find problems with some product of the development process, e.g. code, design.

More systematically seems to work better – don't get sidetracked.

Thus FTR will normally have:

- members who are peers
- designated roles: Moderator, Scribe, Author
- a time limit
- a pre-defined agenda
- a set way to record defects
- no discussion on how to fix defects
- well-defined followup procedure with time limit.

Slide 48

Controversy 1

Some (notably Tom Gilb with Inspection Method) emphasise

- measurement, e.g. of how much time everyone spends, how many defects are found;
- formality, e.g. with written rules defining what a defect is.
- entry and exit criteria: what must be true before an artefact can be reviewed, and what must be true after the review is finished?
- author not allowed to defend own work: only there th answer questions.

Reference: Tob Gilb and Dorothy Graham's book "Software Inspections", Addison Wesley 1993

Slide 49

Controversy 2

But this can take *enormous* amounts of time, not always good value. Others use a more stream-lined approach.

Harmon at IBM, 1991 considered their formal inspections over-attended and over-documented; “interfering with the natural pace of programming”. He made inspections leaner, e.g. got programmers to present their own code.

Success: systems produced on time and still with few defects.

But at the time IBM was under contract to the FAA, which held IBM in breach of contract because they'd changed the agreed process.

IBM apologised and Harmon was reprimanded.

(Story from “Software Runaways”, Robert Glass, pp75 -77)

Slide 50

Controversy 3

In the middle, many companies use some ideas from FTRs informally: e.g.

> [we do reviews] with whiteboard, for the really difficult bits, together with report forms and signoff. All new code and code fixes get reviewed by someone else - usually a peer - before commitment. [...]
>
> Takes ages, and is usually worth it.

(personal communication from s/w eng in generalist IT consultancy)

Slide 51

As the Moderator of an inspection, how can you influence its success?

Slide 52

What might suitable entry criteria for a design document be? And for code?

Slide 53

What advantages do code reviews have over code testing?
Disadvantages?

Slide 54

Why/when might you (not) want a customer representative at a design review?

Slide 55