Verification and validation

Much of the material we need is in **Using UML Chapter 19**, which you should read at this point.

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[1] 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 45

[1]Here I am assuming that the notion of association we choose is the dynamic one: see the book's web page, note on Associations, if you want the precise full story.

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 46

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 47

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 48

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 49

Test automation

Three main benefits from test automation (classification is from *The Automated Testing Handbook* by Linda Hayes, extracts reproduced in *Software Test Automation: Effective use of test execution tools* by Mark Fewster and Dorothy Graham (Addison Wesley 1999)).

- repeatability: obviously you get time savings and increased predictability – provided you can maintain the automated tests...
- leverage: if testing is quicker because of automation you can do more tests than you otherwise would;
- accumulation: older software needs more testing, because of feature accumulation and the need to prevent old bugs reoccurring. Automation can help prevent this being a runaway cost.

The rest of this section is a much abbreviated summary of Chapter 21, by Lloyd Roden, of *Software Test Automation*.

The company concerned, Peterborough Software, specialises in Human Resources software. In 1996 its development teams used a tool called Visual Test but its Independent Test Unit was not happy that that was the best available. They chose a competitor, QARun, after evaluating their requirements of the tool (ease of use was no. 1) and the vendor (future development of the tool in line with PS's products was no. 1). Roden gives as the key issues to note in the evaluation process:

- Produce a requirement list and prioritize which is most important;
- inform each of the vendors that there are others in the evaluation process;
- invite the vendor to demonstate their tool on your own software;
- don't spend too long evaluating the tools;
- once you have decided on the preferred tool, produce a good business case and present it to senior management.

They bought 3 licences and got the vendor to run an in-house training course for the 3 team members, based on their own software. Team members were to use the tool for real immediately after the training (important as you forget what you don't use).

They regarded it as important to have naming conventions and standards (about e.g what test scripts were called and how they were written) and one of the 3 people took ownership of these standards, became the local expert in QARun and generally acted as champion for the tool. (General point: any process improvement needs a champion to drive it through the organisation. In some cases *two* champions are needed: one at the management level to convince everyone of the benefits, and one at the technical level to convince everyone it's doable. This seems to be the case here: the chapter author was acting as management champion.)

They tried hard to keep things simple (the KISS approach). In this case this meant starting with automating very basic "breadth" tests, e.g. to check that all menus do something and catch changes in window's text. Later they moved on to automating more tests. However, it wasn't sensible to automate everything: about 40% of their tests still included manual intervention. The author says it saved them time in the long run to take time to develop generic data driven test scripts rather than hard-coding the test data into the scripts. The tool allowed them to build performance tests easily based on their simple initial tests.

PS found some problems, for example that it takes a lot of effort to maintain test scripts (they're closely coupled with details of the product, so e.g. the switch from VB4 to VB5 was painful). For the same reason they reckon it's important not to start automating systems tests until the software to be tested is robust (!! but this makes sense considering that test automation is most useful for *regression* testing)