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 50

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 51

Formal technical reviews

See also Using UML Chapter 19.

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 52

Important point I forgot to put on the slide: the participants prepare beforehand e.g. by carefully reading the document, code etc. they are to review and making their own notes.

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 53

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 54

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 55

Exactly what's done depends on the company's QMS, but a common version is to review each document produced as part of the development process: so there will be a review of the architectural design document, of the detailed design, of the test plan, etc. Few companies formally review all code, but some will have code reviews for particularly complex pieces of code, or perhaps for randomly selected pieces of code to check overall coding quality.

A design review is not to be confused with a design meeting. There may be many design meetings, which aim to do the design. After the design is completed, a (single) design review checks what has been produced.

A few notes on the questions you considered in groups:

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

Apply rules sensibly: understand their interpretations beforehand.

If it is your job to choose the participants – it often is – choose a mixture of people with appropriate skills and non-conflicting personalitites. Prepare them properly: make sure they understand how the review works and that they should review the document beforehand.

Keep the meeting on track – cut off discussion when necessary.

Slide 56

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

For example:

Design: should be complete with whatever the QMS requires to be present; for example, this might include UML models of the proposed design. Might require certain sanity checks to have been passed, e.g. the kind of checks Rose can do that all classes mentioned are defined.

Code: should compile without error (and maybe without warnings, depending on language and QMS details), should run (but probably won't be fully tested at this stage).

Slide 57

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

Advantages: can assess aspects of quality of code that aren't testable, e.g. adherence to coding standards, quality of comments. May be able to catch certain errors that are hard to catch by testing; e.g. things that go wrong under wierd circumstances, memory leaks.

Disadvantages: not as repeatable as testing – may be hard to make sure that defects which are found stay fixed. Expensive. May be easier to find trivial violations of standards than actual wrong behaviour.

Slide 58

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

Reasons for wanting it: for a requirements document to increase confidence (yours and customer's!) in correctness of requirements. For a design review? Defects in design often correspond to misunderstandings of requirements which may require customer input.

Reasons for not wanting it: issues probably outside customer's expertise or concern.

It may well depend on the type of review: formal reviews require participants to be trained in the format of the meeting, and having a customer in such a thing would be likely to be disruptive.

Slide 59

Management

Main source of material for this section is Using UML Chapter 20.

Management

Two types of management are relevant to software:

- people management
- project management

Both types are best seen as enabling activities: a good manager doesn't do the actual work of building software but aims to maintain an environment in which it's possible for people to get on and do so!

Of course many would argue that the bottom line is always financial.

A manager is not the same as a leader...

Slide 60

One view: management as enabling activity

> No project can succeed without management support. The best sort of management support is the kind in which management doesn't find out about the project until it's a market success. If management notices a project too soon it'll support it in the following ways:
>
> - Demand frequent status reports to explain why the team doesn't have enough time to meet deadlines.
> - Demand explanations of how the project differs from all the projects that have similar acronyms.
> - Ask the team what it could do if it had only half as much funding.
> - Appoint an Oversight Committee whose members are always on trips.
>
> To put it another way, managers understand that their role is to remove obstacles from the project team. They probably could do that, with the help of Dr Kevorkian, but most managers are not such good sports. Therefore, coincidentally, the biggest obstacle to the success of any project is management itself.

(from The Dilbert Principle, p233)

This is of course a caricature: but the kernel of fairly uncontroversial truth here is that an important function of the manager of a task is to keep mindless bureaucracy away from the people who are doing the task, maybe by making the necessary arguments to make it go away, and if necessary by doing the dirty work him or herself.

Another view: management as financial control Here's an extreme expression of a financial view of management (from Drucker, P.F. The Practice of Management):

> Management must always, in every decision and action, put economic performance first. It can only justify its existence and its authority by the economic results it produces. There may be great non-economic results: the happiness of the members of the enterprise, the contribution to the welfare or culture of the community, etc. Yet management has failed if it fails to produce economic results. It has failed if it does not improve or at least maintain the wealth producing capacity of the economic resources entrusted to it... The first definition of management is therefore that it is an economic organ of an industrial society. Every act, every decision, every deliberation of management has as its first dimension an economic dimension.

Of course, we may need to take a very wide view of economics to make this view defensible: the manager of a safety critical software project cannot ignore the safety of human beings, but maybe that can be accounted for by the effect of a failure on the financial position of the company. On the whole I tend to think not...

Project management

Software project management

A project manager arranges for the following functions to be fulfilled:

- Planning
- Organizing
- Staffing
- Monitoring
- Controlling
- Innovating
- Representing

Slide 61

(see also the longer list of functions in Chap 20.)

Some of these (most of) you have covered in considerable detail in PM: risk management, planning, estimation and scheduling for example. Here I want to emphasise a couple of other aspects.

How does this interact with QA? The QMS should lay down how the organisation ensures that these activities are done competently. For example, it might specify what information should be included in a project plan, and what the procedures for signoff of the plan are (this might include having a review meeting and getting the customer's signature, for example).

Here's one possible outline of project plan, from OU M355 Unit 13, at the kind of level that a QMS might prescribe:

1. The Project Introduction
    - project plan outline

- any assumptions made when plan was produced
- project schedule

2. The Project Description
   - project products
   - acceptance criteria for products
   - customer involvement in development tasks
3. The Phase Plan
   - project phases and associated tasks
   - project milestones and associated dates
4. The Organization Plan
   - staff and their responsibilities
5. The Support Requirements Specification
   - software development facilities required
   - hardware development facilities required
   - support required for the software development team
6. The Development Process Specification
   - development techniques and standards to be used
7. The Change Control Plan
   - method for handling changes during the project
8. The Documentation Plan
   - methods of document production
   - document format
9. Appendices
   - test plan
   - quality plan
   - example development notations

Notice that the project plan is a customer-readable document. This has several implications.

- It must be comprehensible by the customer. (Hence, for example, the possible inclusion of example development notations in an appendix. The developers are presumably familiar with the notations they use, but the customer will probably not be.)
- It must show the developers in a good light: any problems you highlight will come with suggested solutions, unless you can argue that none exist. It will not mention risks such as "we don't have the expertise to do this"!
- It must not include business-sensitive information you don't want the customer to know, e.g. cost estimate, as distinct from price estimate – you don't tell the customer your profit margin...

Different criteria might apply if you were planning an in-house project – but increasingly IT divisions bid against outside suppliers, so the gap is not as clear as it was.

Managing teams Required reading: Project Risk Reduction Patterns by Alistair Cockburn. The details are not examinable, but the essence is: that is, I wouldn't ask a question like "What is the Gold Rush pattern?" but I *could* describe a problem situation like the ones addressed by these patterns and ask you to make some sensible suggestions for solving the problem.

There are two main reasons why I've chosen to hand this out:

- It illustrates the way that quite small-scale changes in the ways things are organised can make a big difference to the success of a project as it goes on. The patterns in this catalog might be of use to a team leader, not just the overall project manager. This seems particularly relevant as team leading is something you may well have to do soon, whereas overall project management for large projects is further in the future.
- We covered patterns for design earlier in the course. Here we see patterns used to convey solutions to management problems.

Project tracking

As the project goes on, estimates have to be revised in the light of progress, unforeseen problems etc. Typically a large project will replan once a week.

Depending on the circumstances slippages may mean

- reallocating resource (but see The Mythical Man Month)
- cutting functionality
- asking the customer for more money
- losing profit

Tell them the most important metric is what proportion of the project staff's time is spent explaining to the customer why the project is late Anonymous software engineer.

But why do projects always slip?

And how do we track the project?

Slide 62

Why do projects always slip?

Discussed in (optional but rather interesting reading):

The Rational Planning of (Software) Projects

by Mark C. Paulk.

This paper discusses the effects of three features of human nature:

- people tend to be risk-averse when there is a potential of loss
- people are unduly optimistic in their plans and forecasts
- people prefer to use intuitive judgement rather than (quantitative) models

It goes on to discuss how a framework like the CMM can help.

Slide 63

Metrics

You discussed metrics in PM, mostly technically, and mentioned that they are controversial. One of the main reasons for this is that if you don't have the right metrics, they are useless.

Use of metrics

You've encountered metrics before. But how do we decide which metrics are useful? A metric may be a:

- predictor – use one measurement (e.g. number of functions) to predict another (e.g. cost of project). The thing used is a predictor metric.
- result – if a measurement was predicted by a predictor metric, measure its actual value at the end of a project (or phase). Feed the discrepancy back into process improvement and future estimation.

For example, by tracking metrics through the project, a manager can follow the progress of the project compared to what was planned.

Slide 64

Useful metrics must be...

- Measurable – e.g. not someone's opinion of how complex something was
- Independent – i.e. not something that can be altered without altering the quality of the software
- Accountable – i.e. managers should be able to justify both the use of this metric and the value given to it.
- Precise – i.e. the minimal accuracy of the number must be known.
- MEANINGFUL!! – there must be some reason to believe that numbers for the metric have something to do with software quality!

Slide 65

You may think that there are no such things as truly independent metrics and, indeed, they are rare. The reason that you seldom encounter independent metrics has to do with *Heisenberg's uncertainty principle (slightly revised).*
Measuring any project parameter and attaching evident significance to it will affect the usefulness of that parameter.
The keyword here is *evident.* If the project members know that you can count the number of smiles on Monday mornings and use that as an indication of project morale, then when they want you to believe that morale is high, they will all start smiling on Monday mornings. If on the other hand you keep to yourself the metrics you are using and the significance you attact to them, their usefulness (if any) will not degrade over time.

DeMarco, T. (1982), *Controlling Software Projects*, Yourdan Press, New York

To justify the metric – to make it accountable – will have to store:

- exactly what was measured
- by whom
- when
- what values were for other projects

It is essential, and can be hard, to measure the things you want to know. However software metric expert Tom Gilb propounded what Tom DeMarco dubs Gilb's law:

Anything you need to quantify can be measured in some way which is superior to not measuring it at all

Is there a contradiction here?

People management

Line management

As a software engineer, you probably at any one time have both

- a project manager (determined by what project you're on)
- a line manager (determined by who you are)

Your line manager oversees your career progression, training, overall performance, and probably has a role in assigning you to projects.

These managers may or may not (esp. *matrix management* be the same person. Should they be?

Conflict of interests (so no) ?

Communication problems (so yes) ?

Slide 66

What are the characteristics of a good manager?

Successful teams

What makes a successful team?

- balance of appropriate personal and interpersonal skills
- balance of appropriate technical skills for the task
- people who get on with one another
- being part of a basically happy and successful organisation
- appropriate size (3 - 8 ?)
- good team manager
- low turnover
- success!!

Slide 67

The idea of a balance of personality types, or team roles, has been formalised by Meredith Belbin into the following:

Externally focused/outward looking roles

Chairperson Enjoys formally presiding over the team and ensures co-ordination to meet goals

Plant Seeks to provide new ideas and bring these into solution development

Resource investigator Goes outside the formal team structure to find knowledge for the team to use

Shaper Adopts a results oriented approach and seeks to reach decisions and take action

paragraphInternally focused/inward looking roles

Company worker Provides organisation and focuses on the practicalities of what need to be done

Monitor evaluator Evaluates suggestions and proposals that the team is considering

Team worker Seeks to build the team through listening and providing suggestions on team effectiveness

Completer finisher Focuses on task completion by ensuring that sufficient details exist to support actions and key deliverables

Each role has advantages and disadvantages – a sucessful team has a balance. Individuals don't of course always play the same role but people tent to have some they frequently play and others they never play. What are yours?

Leadership

A leader is

- one who makes things happen
- a shaker-up
- a vision person
- someone who sees what's important
- not easily diverted from his/her course

Can someone simultaneously lead and manage?

Slide 68

What makes a good leader?
Do you think you are more likely to make a good manager, or a good leader?