University of Dublin
Trinity College

3BA7: Software Engineering

René Meier
Rene.Meier@cs.tcd.ie
http://www.cs.tcd.ie/Rene.Meier

Structure

Hilary term

- Software lifecycles
- Domain analysis, requirements, specification
- Working in a team
- Project planning and management

Trinity term

- Course work
- Project planning and management
- Design issues
- Testing and implementation
- Configuration management

Structure

Tutorials

- Supplement lecture notes
- Will be assessed in combination with the lecture notes

Seminars

- Invited speaker(s)

Course material and information

- http://www.cs.tcd.ie/Rene.Meier
- Lecture notes will be made available shortly after lectures

Course work

One major group project – building software systems

- Groups of 5/6, group members will be assigned
- Covering numerous aspects of building software systems

- Usage of development tools
- Software design and implementation
- Project team management
- Project management

- Several deliverables, each of which will be assessed!
- Presentations and demonstrations

- Starts during week 5 of this course
- See course webpage

Questions?

"Software engineering"

What is it and why do we need it?

- Typical "student" project: implementing a "Message Queue" in Java
- Engineering analogy: building a piece of D.I.Y., for example a book shelf
- How would you approach these problems? What are the problem solving "stages"?

An engineering analogy

Building a piece of D.I.Y.
versus
Building a bridge

Up to now, the programs you've written are probably more like D.I.Y.

Software engineering is more like building (or architecting) bridges

The software engineer's tools…

"Software engineering provides us with structures and techniques that make it easier to handle complexity."

Forget all this small-scale stuff…

Large projects

- Systems with several million lines of code aren't uncommon
- Project development times in person-centuries
- Anticipated project lifetimes often run to decades
- Real lifetimes are indeterminate – some systems *can't* die

No one person knows the whole system intimately

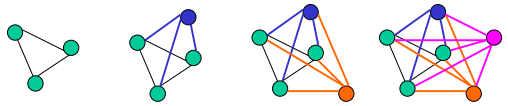
- Communication, compromise

Not just a small system with more code

The purpose of this course is to teach you about large-scale software and how to build it successfully

The first basic problem

…is one of communication – both personal and in the software



3 agents 3 channels | 4 agents 6 channels | 5 agents 10 channels | 6 agents 15 channels

A system "twice as big" is actually much more than twice as complex

How do we master this complexity as systems grow?

The second basic problem

…concerns how systems evolve and goalposts move

A system which doesn't change isn't being used

- New platforms, new peripherals, new modes of working
- The web changed every application – and the companies that didn't notice watched their businesses die

But users want a certain amount of stability

- Investment in training and support – may be more than in software
- Resist moving to Linux because of the re-training – even if it offers a better technical solution to their problem

How do we balance this trade-off as systems grow?

What is "software engineering"?

"The art, craft and science of building large, important software systems"

Software systems

A software system isn't just software

Key tasks

- What to build?
- What business must it fit into?
- What to run it on?
- How to build it?
- Does it work?
- How do people use it?
- Can they use it advantageously?
- …and then keep it running for the (un)foreseeable future

Engineering systems

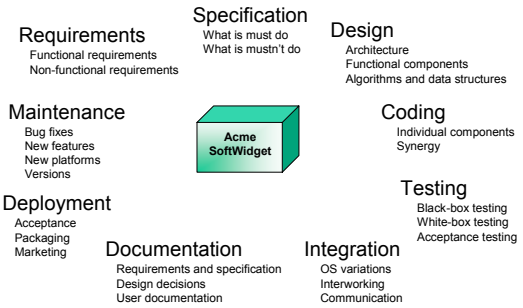
"The application of science to the design, building and use of machines, constructions etc"

Oxford Reference English dictionary (1998)

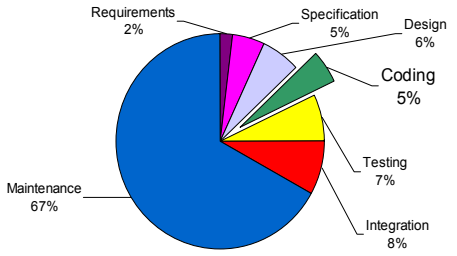
Implications

- Repeatable – should be able to re-use tools and techniques across projects
- Responsible – should take account of best practices and ethics
- Systematic – should be planned, documented and analysed
- Measurable – should have objective support both for the products *and* for the process itself

What's involved



How long each bit takes



From Stephen Schach, *Classical and object-oriented software engineering*, Addison-Wesley (1999)

For example

Consider a space mission

• On the ground under construction	3 years
• In flight	3 years
• Data collection	5 years
• Data analysis	10 years
• Total	21 years

Of that

- 50% maintaining the results, keeping them accessible
- 35% maintaining a running system – at a distance
- 15% analysis/design/coding/testing

What this means

A large project is basically all maintenance

- Coding is an almost insignificant fraction
- Spend as much time designing as coding
- Testing and integration are hard

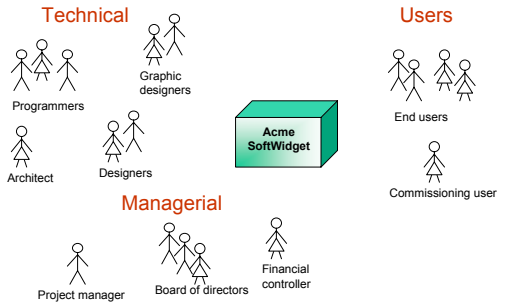
Any system which doesn't change isn't being used

- A useful system is *never* complete
- …and will get used in ways you never imagined

Plan for change

- In requirements, platforms, user interfaces, …
- Make the system maintainable from the start

Who's involved



The people

Always both the strongest and weakest element

Many (most?) will not be very technically literate

- How do they tell you what's wrong?
- How to you tell them what you're doing?

In most organisations, programmers do not make any decisions beyond the small-scale software

- Part of being an engineer is to be able to deal with, and communicate with, a range of people so as to get the job done right and on time
- Part of being a manager is to listen to them, when they know the technical issues better than you do

The importance of being “important”

An important system can’t “just fail”

- Safety-critical, mission-critical, …

Far more time in maintenance than in development

Why you can still make a good living as a COBOL programmer

- Applications must be kept running no matter what the cost
- Archaic software hangs around
- Can’t always re-engineer to the “correct” solution