UCL

COMP2009

Introduction to Software Engineering

Dr. Graham Roberts
Dept. of Computer Science
http://www.cs.ucl.ac.uk/staff/g.roberts
g.roberts@cs.ucl.ac.uk
Departmental Tutor
Director of Studies

UCL

Content Overview

- Primarily about Object-Oriented (OO) modelling
  - Constructing requirements and use cases.
  - Analysing requirements to create an OO model.
  - Refining model to create a system design.
  - UML notation and diagrams.
  - Understanding core OO concepts.
  - Methodology, in particular USDP.
  - Plus some general SE.
- Why?
  - OO + UML are widely used.
  - Need to know how to describe requirements and model a system.

UCL

Moodle Reminder

- Locate Engineering Sciences > Computer Science
- Enrol on:
  - COMP2009SE: Software Engineering
    - Enrolment key is: uml2

UCL

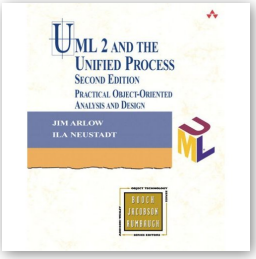
Coursework

- 3 binary marked courseworks
  - 1% on domain model + glossary
  - 2.6% on requirements + use cases
  - 3% on OO model
- To be submitted online via Moodle.
- Feedback on Moodle via common mistakes list and examples.
- This is formative coursework.
  - You are practicing how to do various tasks.
  - Take it seriously otherwise you will not gain any experience.

UCL

Reminder: Compulsory Course Text (SE part)

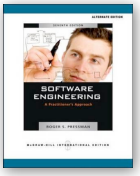
UML 2 and the Unified Process
by Jim Arlow and Ila Neustadt,
pub. by Addison Wesley, 2005
ISBN 0321321278
(978-0321321275)



You *must* read and study this text in detail. Assessment will assume you have. I won't cover everything in lectures.

UCL

General SE Books



Software Engineering: A Practitioner's Approach 7th ed. by Roger S. Pressman, published by McGraw-Hill, 2009, 0071267824

Software Engineering 8th ed. by Ian Sommerville, published by Addison Wesley, 2006, 0321313798

Both good general SE texts. Recommended for background reading.

Other Relevant Books

- The Mythical Man Month, Frederick P. Brooks Jr., Addison Wesley, 1995, 0201835959
  - A classic, everyone assumes you have read this!
- UML Distilled, Martin Fowler, Addison Wesley, 2003, 0321193687
- Writing Effective Use Cases, Alistair Cockburn, Addison Wesley, 2000, 0201702258
- Extreme Programming Explained: Embrace Change, Kent Beck, Addison Wesley, 2004, 0321278658

Online resources - ACM, IEEE

- Goto http://www.ucl.ac.uk/Library/ejournal/ejcompsci.php
- ACM Transactions and Digital Library via portal at http://www.acm.org, http://portal.acm.org/portal.cfm
  - ACM Transactions on Software Engineering and Methodology
  - and large number of other journals
- IEEE Computer Society http://www.computer.org
  - IEEE Transactions on Software Engineering
  - Software Engineering Journal
- Plus publications from Wiley, Springer and many more.

Big conferences

- OOPSLA (Object-Oriented Programming Systems, Languages and Applications)
- ECOOP (European Conference on Object-Oriented Programming)
- ICSE (International Conference on Software Engineering)
  - Visit web sites to find out more.

What else?

- Time spent reading is worthwhile.
  - Check prices on Amazon.
- Also use web-based material.
  - Wikipedia is a useful starting point.
    - Good SE content but check sources.
  - uml.org, official OMG website.
    - Object Management Group, maintains and develops UML standard.

What is Software Engineering (SE) ?

Wikipedia says:

- "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software".
- "an engineering discipline that is concerned with all aspects of software production"
- "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines"

Design and Engineering

- Engineering is
  - the application of *scientific* principles and methods to the construction of *useful* structures or machines.
- Design is
  - the essence of any engineering discipline
  - the *creative* selection of appropriate principles and methods to efficiently and effectively *transform* concept into reality
- A large part of software development is really about design in various forms.

SE Origins

- The idea and the label are nearly 40 years old.
  - First used at NATO Conference on Software Engineering, Garmisch-Partenkirchen, Germany, 1968
- "Software Engineering" deliberately chosen as provocative ... an aspiration.
  - "Expressed a need rather than a reality."
- Serious problems with software development needed to be solved scientifically and rigourously.
  - Ironically many of the same problems remain today...
- By the time of the conference the following year in Rome, the label "Software Engineering" had stuck fast even if no one knew how to achieve the goals.
  - Still don't really know...

Present day reality

- The reality is mixed
  - Computer Science is the scientific basis.
  - A growing number of software development approaches are recognised as good engineering practice.
  - But many aspects of development are still ad hoc rather than scientific.
- Software design is still very difficult.
  - Few guiding scientific principles.
  - Few universally applicable methods.
  - Much poor practice.
  - Frequent failures.

Key Issues

- Software Engineering principles are particularly relevant to construction of *large systems*.
  - Involving large numbers of people to manage.
- Mastering complexity.
  - Abstraction, information hiding, separation of concerns.
- Continual evolution.
  - For improved reliability, incorporation of new features, improving other desirable qualities.
- Efficiency of development.
  - Human-centered tasks are extremely costly.
- Cooperation of people.
  - Cannot develop a system in isolated pieces.
- Usability.

Properties of Good Software

- Delivers required functionality.
- Maintainable
  - can evolve to meeting changes in requirements.
- Dependable
  - Robust, reliable, trustworthy.
- Efficiency
  - Good use of resources: computational, user time, development time/cost
- Usability
  - Usable by the users (or systems) it is designed to interact with.

Scale

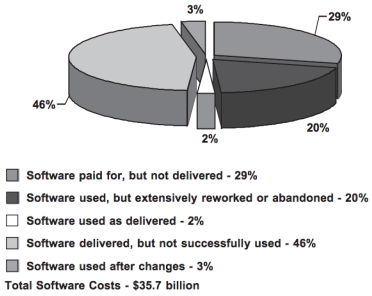
- Scaling-up does not work reliably
  - not easily understood by one person
  - communication overhead
  - effect of changes not obvious
  - need for discipline, documentation and management

Note:
It is very important that you keep the problems of scale and complexity firmly in mind throughout the course.

Failure...

- Numerous examples of failed or seriously delayed software development projects.
  - Or fail to deliver full functionality within time and budget.
  - E.g., Pensions system, NHS patient records, Taurus (Stock Exchange), Air Traffic Control
  - Disasters: ESA Arianne 5, Patriot Missile System, Therac-25 radiation therapy machine
  - Just about every UK government software development project...

Scale of the problem...



The Mythical Man-Month
Frederick P. Brooks, Jr.

- Published 1975, Republished 1995 (Addison Wesley)
- Experience managing the development of OS/360 in 1964–65
- Central Arguments
  - Large programming projects suffer management problems different in kind from small projects, due to division of labour.
  - Critical need is the preservation of the conceptual integrity of the product itself.
- Central Conclusions
  - Conceptual integrity is achieved through exceptional designers.
  - Implementation is achieved through well-managed effort.

No Silver Bullet
Frederick P. Brooks, Jr.

- "No Silver Bullet—Essence and Accidents of Software Engineering", Computer, April 1987

  There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

- Essence—"the difficulties inherent in the nature of the software" (complexity)
- Accidents—"those difficulties that today attend its production but that are not inherent" (technology)
- Solution: Grow Great Designers

This Module

- Can only cover a small sub-set of SE.
- So will focus on analysis and design using UML.
- Why?
  - Object-Oriented Development and UML very widely used.
  - Directly relevant career knowledge/skills.

The Software Development Process

- Structured set of activities required to develop a software system, including:
  - Requirements Gathering
  - Analysis
  - Design
  - Implementation
  - Testing
  - Deployment
  - Maintenance
- Activities vary depending on organization and type of system being developed.
- Must be *modelled* in order to be *managed.*

Requirements Gathering

- Aim: a complete description of the problem and of the constraints imposed by/on the environment
- Description may contain:
  - Functions of the system
  - Future extensions
  - Amount of documentation required
  - Response time and performance
  - Acceptance criteria
- Result: Requirements Specification

Analysis

- Aim: Analyse requirements to create a *conceptual model* of the software system.
  - Data modelling
  - Functional modelling and information/control flow
  - Behavioural modelling and state
  - User interface modelling
- Result: A set of Analysis Models

What is a model? A systematic conceptual description of a system, or aspect of a system, defining its structure and behaviour. Abstraction and simplification are used to constrain the complexity while retaining the essential information.

Design

- Aim: an implementable model of the software system.
  - Sufficient information to allow system to be built.
- Architecture is defined.
  - The definition of the global architecture of the system is essential.
- System is decomposed to components within the architecture:
  - Definition of component interfaces and functionalities
- Design decisions dramatically impact system quality.
  - Wrong architecture/design == throw it away.
    - You can't build a tower block on the foundations for a bungalow.
- Result: Detailed Design Documentation

Implementation

- Aim: implementation of all design elements.
- Starts from the component specifications developed during design.
  - Interfaces defined in the design should be respected by the implementation of the component.
- Code should be well documented and easy to read, flexible, correct, reliable AND fully tested.
- Result: working software.

Testing

- Kinds of testing:
  - Unit testing: Classes/methods.
  - Functional testing: Component implementations against their interfaces.
  - Integration testing: Component integration against system architecture.
  - System testing: validation of the extent to which the requirements specification is fulfilled.
  - Acceptance testing: validation of external behaviour against user expectations.
  - Testing and implementation should run in parallel.
- Result: Fully tested software

Deployment

- Package software ready to install on a computer system.
- Actually installing the software.
- Live testing in real environment.
- Documentation and manuals.
- Training.
- Result: Working software in situ

Maintenance

- Aim: keeping the system operational after delivery to the customer.
  - Corrective: identification and removal of faults.
  - Adaptive: modifications needed to achieve interoperability with other systems and platforms.
  - Perfective: incorporation of new features, improved performance and other modifications to increase customer satisfaction.
  - Preventive: modifications to mitigate possible degradation of usability, maintainability, etc.
- The maintenance phase needs to be considered part of the overall development process.
  - Even though the activities within maintenance are requirements analysis, design, implementation and testing.
  - Maintenance is ongoing and expensive.

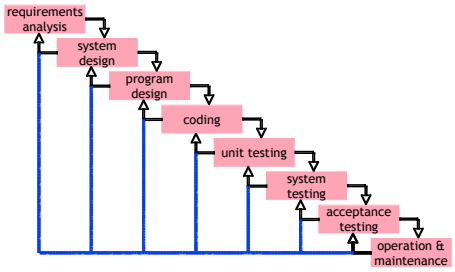
Process Model

- A development process is described in a *process model*.
  - There are many ways of organising development.
  - There are many process models described in the literature.
- You need to model the process because:
  - when a team writes down a description of its development process, it forms a common understanding of the activities, resources and constraints involved in software development..
  - creating a process model helps the team find inconsistencies, redundancies and omissions in the process; as these problems are noted and corrected the process becomes more effective.
  - the model reflects the goals of development and shows explicitly how the product characteristics are to be achieved.
  - each development is different, and a process has to be tailored for different situations, and so the model helps people to understand these differences.

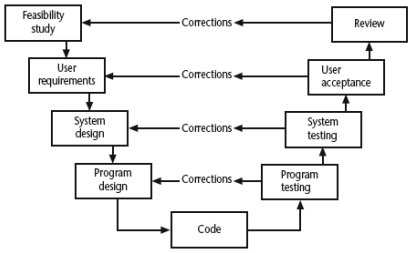
Common Software Lifecycle Models

- Waterfall Development
  - Separate and distinct phases of specification and development.
- Prototyping
  - Interleaved specification and development.
- Incremental Development
  - Development of a system in increments.
- Agile Development
  - Continual availability of a fully tested functioning system, developed with minimal organisational overhead.
- Ad Hoc

Waterfall Model (Royce 1970)

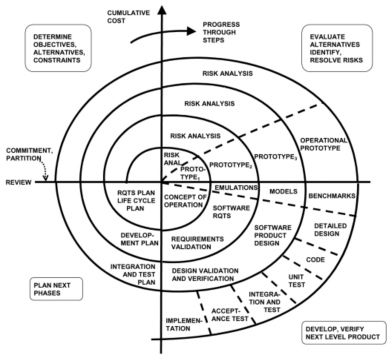


V Model

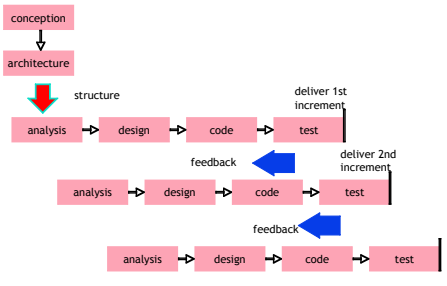


The Waterfall where each stage on the left has a testing (validation) process on the right to detect feedback and a loop back path to allow reworking.

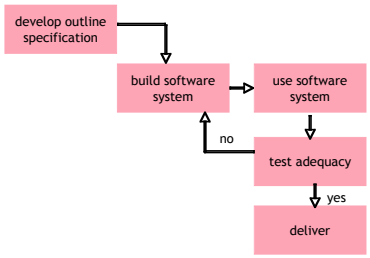
Spiral Model (Boehm, 1988)



Incremental Model

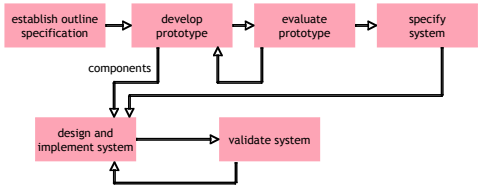


Exploratory Model



Incrementally build a system using test/user feedback to guide development.

Prototying Model



Construct a *prototype* to determine how to specify and build the real system.
The prototype implementation should not be used as the real system.

No Model

Muddle through and hope for the best...

Summary

- Overview of what SE is.
- Core development activities.
- Introduction to Lifecycle Models.

- Use SE text books and online sources like Wikipedia to find out more.