CS4507 Advanced Software Engineering

Lecturer: Adrian O'Riordan
Office: Room G.71 WGB
Email: a.oriordan <at>cs.ucc.ie

Course Webpage: http://www.cs.ucc.ie/~adrian/cs4507.html

CS4507 Overview

5 Credit course on Software Engineering (upper undergraduate level)

- Pre-requisite: *CS3500 (Software Engineering)* or equivalent
- Lectures: 2 lectures per week in Terms 1 – Tuesdays 9-10am (G.04 WGB) and Thursdays 12-1pm (G.02 WGB)
- Labs: to be announced, starting in week 5 (G.21 WGB)
- Tutorials: as required

Assessment will consist of an end-of-year written examination (80%) and continuous assessment during the year (20%).

You have to pass combined total. There is a re-sit in the autumn – your continuous assessment mark is carried forward.

CS4507 On-line

Webpage at http://www.cs.ucc.ie/~adrian/cs4507.html

Will contain:

- Course Overview: module content, *etc.*
- Notices
- All lectures slides (as course progresses)
- Reading list and Web links
- Assignments and Exercises

CS4507 Learning Outcomes

According to Book of Modules:

- Participate in all development activities of a software engineering project;
- ~~Evaluate the management of a software project;~~
- Use a modelling language, such as the Universal Modelling language (UML);
- Follow a model driven software development process;
- Use important software design patterns;
- Develop working software using a commercial software modelling tool.

Teaching Methods

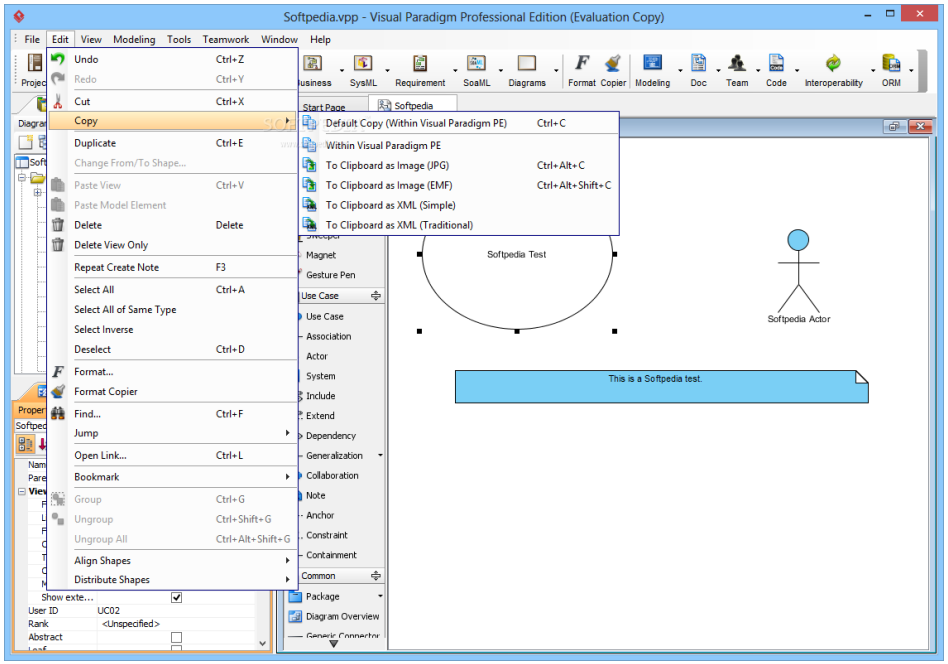
- It is important that you attend both the lectures and labs.
- Assignments and exercises will be placed on the course webpage during the year.
- No textbook covers all the material exactly. See the list of relevant books later on.
- Readings will also be assigned during the year.

Labs

- Labs will run from probably week 5.

- Labs will involve object-oriented analysis and design using the UML CASE tool *Visual Paradigm (formerly Visual Paradigm for UML)*
  - *http://www.visual-paradigm.com/*

- This tool supports
  - Requirements modelling
  - UML Modelling
  - Code generation
  - Report generation
  - Database design
  - Collaborative Design



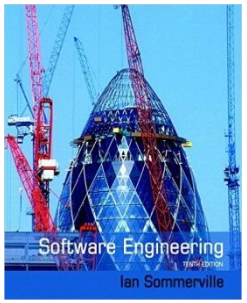
Course Contents

- Software Engineering Overview *(1l)*
  - Software Engineering, History, Disasters
- Process and Lifecycle models *(4l)*
  - Waterfall, Iterative/Incremental, Spiral, RUP, MDA
  - Agile Development: Agile Manifesto, Scrum, XP
- Requirements and Analysis *(2l)*
  - Requirements Specification, Use Cases
- Object-Oriented Design in UML *(10l)*
  - UML notation, subsystem specification
  - Object-Oriented Design in UML: domain models, GRASP patterns, design models, state modelling, implementation
- Software Design *(4l)*
  - Programming Idioms, Design Patterns, Refactoring and Reengineering
- Software Validation and Verification *(2l)*
  - Testing
  - Software Quality Assurance
  - Software metrics, CMM/CMMI

Approximate number of lectures in green

CS4507 Relevant Books and Websites

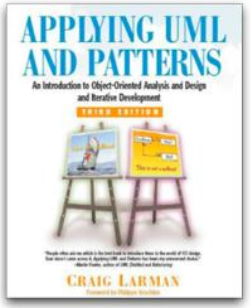
I'll be using notes based on the following two books:



Software Engineering, Ian Sommerville, Pearson, (10th ed. 2015 or 9th ed. 2010).

and

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edition, Craig Larman, Prentice Hall, 2004.



Further reading list on course Website:

http://www.cs.ucc.ie/~adrian/cs4507/CS4507books.html

And relevant Web links

http://www.cs.ucc.ie/~adrian/cs4507/CS4507refs.html

What is Software Engineering?

- Definitions:
  - (1) The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software *(ISO/IEC 2382-1:1993 Information technology--Vocabulary--Part 1: Fundamental terms)*
  - (2) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software *(ISO/IEC/IEEE 24765:2010 Systems and software engineering--Vocabulary)*

- Large-scale software development characterized by
  - project involves a *team* of people, need to manage process, people and artefacts;
  - system takes a long-time to build – need to *plan*;
  - systems *complex* – need powerful tools, methods and technologies ;
  - need to try to *reuse* code/designs/process.

Software Engineering Objectives

- Software Engineering is relatively new field compared to established engineering disciplines such as electrical engineering

- Attributes of “good” software:
  - correct
  - acceptable
  - secure
  - robust
  - maintainable
  - efficient

- Attributes of “good” software development process:
  - disciplined/systematic
  - repeatable
  - quantifiable

A Very Brief History of SE

- The term software engineering can be traced to a 1967 NATO study group and the first conference was held in Germany in 1968 sponsored and facilitated by NATO.
  - Report is online http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

- Concept of *modularity* and *information hiding* introduced
  - Modularity is a design principle for separating a computer program into distinct sections.
  - *Information hiding*: containment of a design or implementation decision in a single module [D. Parnas, 1972, On the Criteria To Be Used in Decomposing Systems into Modules, Comm. ACM, 15(12)] http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf

A Very Brief History of SE *continued*

- Software crisis:
  - “The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! ... programming has become an equally gigantic problem.” Dijkstra in ACM Turing Lecture 1972
  - https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

- In mid-70s demands of the new operating system OS/360 taxed the limits of the programmers, project managers, and the resources of the IBM corporation
  - detailed in classic *The Mythical Man-Month,* Frederick Brooks, 1975
  - Brooks's law: Adding manpower to a late software project makes it later.
  - http://books.cat-v.org/computer-science/mythical-man-month/tmmm.pdf

Ongoing Problems

In 1980s software engineering became established *profession* but ongoing problems:

- software not acceptable to customer or users
- buggy
- behind schedule/over budget
- unmanageable and code difficult to maintain
- software was very inefficient
- not enough reuse – either code or design
- no rigorous measurable process

Software Crisis: No Silver Bullet

- Frederick Brooks in 1986 addressed the issue
  - "there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity.“
  - Brooks identified inherent problems associated with software production which are caused by the nature of software itself called *essential complexity* (unavoidable problems)
  - as opposed to *accidental complexity* (could engineer solutions to these)
  - essential complexity of software due to its complexity, conformity, changeability, and invisibility.
  - "No Silver Bullet — Essence and Accident in Software Engineering". *Proceedings of the IFIP Tenth World Computing Conference*: 1069–1076. http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf

Accidents and Essence

- Accidental difficulties:
  - problems with current/ past/ future technologies
  - problem with the operating systems, compilers, languages, processes

- Essential difficulties:
  - *complexity*: software systems among most complex systems ever created
  - *conformity*: part of larger constraining environment of hardware/users/other software
  - *changeability*: requirements constantly changing
  - *invisibility*: software hard to comprehend and visualize

Software Disasters – Ariane 5

Large-scale disasters attributed to software defects offer sober warnings:

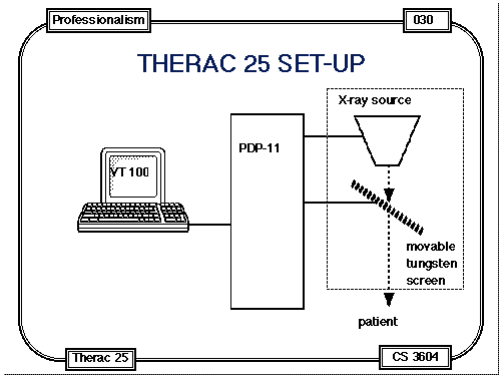
- Explosion of the European Space Agency's Ariane 5 (501) rocket on maiden flight in June 1996
  - exploded 37 seconds after lift-off from French Guiana
  - software error in the inertial reference system (IRS), software to maintain trajectory
  - 64 bit floating point number relating to the horizontal velocity of the rocket converted to 16 bit signed integer number too large thus the conversion failed
  - IRS software reused from Ariane 4 without change, although it included additional functionality that was not required in Ariane 5.
  - unhandled exception (exception handler was not included in the system because it wasn't needed in Ariane 5) caused run-time system to shut down the IRS and launcher stability could not be maintained.

Case Study in Sommerville

Video clip

Software Disasters – more examples

- Therac-25 incident (1985-1987)
  - radiotherapy machine administered lethal doses of radiation to patients
  - concurrent programming errors in software controller



- Patriot missile failure during the 1st Gulf War
  - In Saudi Arabia in 1991 resulting in 28 deaths
  - poor handling of rounding errors

Prognosis

- Reports on major incidents offer analysis of root causes and recommendations, *e.g.*
  - ARIANE 5 Flight 501 Failure Report by the Inquiry Board, 1996
    - found validation tests (which passed) were based on Ariane 5 requirements and not reused software.
  - *An Investigation of the Therac-25 Accidents, IEEE Computer, 1993*

- Some example recommendations:
  - software code should be independently reviewed and include external participants
  - need to verify range of values of variables
  - need high test coverage with realistic input data

- Led to increased support for research on ensuring the reliability of *safety-critical systems*