Comp 30160: Object Oriented Design

Mel Ó Cinnéide
School of Computer Science and Informatics
University College Dublin

Timetable

- Lecture schedule:
  - Tues 3-4
  - Thurs 4-5
- Practical slot
  - Tues 4-6
  - There won't be a lab every week
- When there isn't a lab on Tuesday, I may use the lab slot for lectures in place of Thursday's lecture slot!
- Teaching Assistant: Erika Duriakova (erika.duriakova@ucdconnect.ie)
  - Contact Erika with any questions/issues you have regarding the labs

Course Overview

- This course examines issues in Software Design.
  - OOD + other stuff
- Essential Prerequisites
  - A working knowledge of object-oriented programming
  - Proficiency in Java
  - (For example, a knowledge of the material of Comp30070 is a sound basis for this module)
- Practical work
  - 6-8 laboratory sessions in this module. Indicative topics are:
    - Agile Methods
    - Test-Driven Development
    - OOD with UML
    - Code Smells
    - Refactoring
    - Software Design Principles
    - Design Patterns

Module Topics

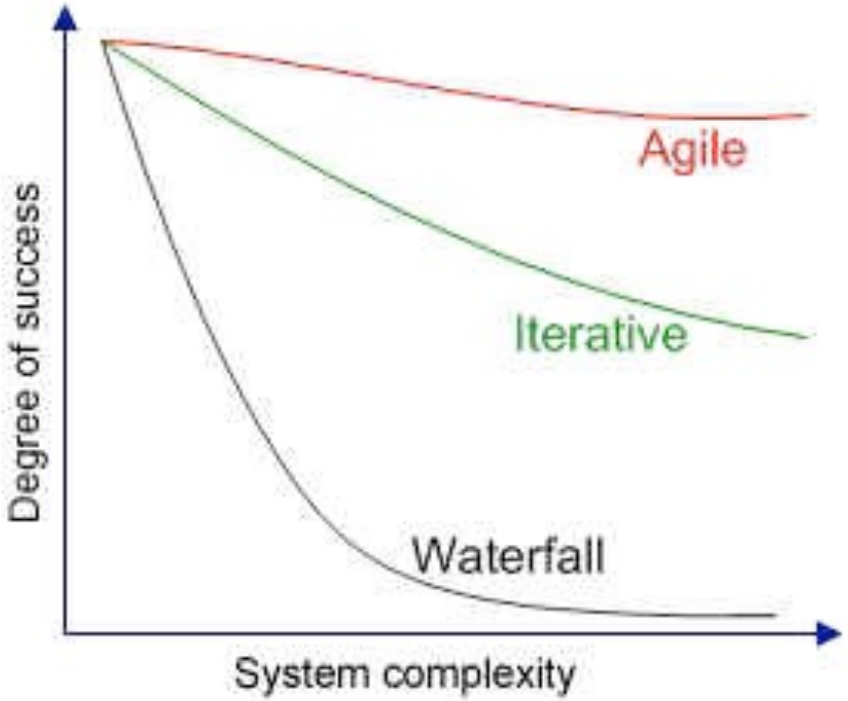
- The module involves a number of topics
  - Software Methodology
  - Object-Oriented Design using UML
  - Software Testing
  - Design Principles
  - Code Smells and Metrics
  - Refactoring
  - Design Patterns

- We'll review each of these topics on the following slides...

Software Methodology

- The process of developing software has evolved enormously in the past half century (or even decade) and remains a fluid, complex area.

- We'll look at two development processes in more detail
  - Unified Process
  - Agile Processes



OOD and UML

- The standard notation for expressing an object-oriented design is **U**nified **M**odelling **L**anguage or **UML**.
- We'll see how various aspects of a system can be modelled using UML
  - Little emphasis on the notation itself; I assume this is familiar to you (or easy to pick up).
- One topic is to examine how a UML design can be used to drive the construction of the code.

Testing

- Of all software qualities, **correctness** is the most fundamental.
  - a *sine qua non* property
- The main tool for achieving and maintaining correctness is **testing**.
  - Unit testing
  - Integration testing
  - Acceptance testing
- We'll see that testing is not only related to correctness, but also to design quality improvement.

Design Principles

- What is good design?
- What is bad design?
- As software engineers, should we care?
  - Usually, yes!

- In this section we examine some of the fundamental principles that underpin software design
  - **S**ingle Responsibility Principle
  - **O**pen Closed Principle
  - **L**iskov Substitution Principle
  - **I**nterface Segregation Principle
  - **D**ependency Inversion
  - Law of Demeter
  - ...

- We'll that these design principles are not just notional elegance, they impact directly on vital qualities like **maintainability**.

Code Smells

- How can we tell if the design of our software has problems?
  - *Code Smells* are one approach
- A Code Smell is a sign in our software that all is not well
  - e.g. Long Method
- Smelly code will exhibit some these properties
  - Hard to understand
  - Hard to update
  - When updated, will lead to more smelly code
- In this section we'll look at a number of well-known code smells
  - Long Method
  - God class
  - Feature Envy
  - ...

Refactoring

- What do we do with smelly, unmaintainable code?
- **Refactoring** is the process of improving a program's design, without changing its behaviour.
  - E.g., a **long method** might be broken into several, smaller more cohesive methods.
- How can we be sure that program behaviour hasn't been changed?
  - We can use a refactoring tool that promises this (not 100% reliable!)
  - Use testing to ensure that behaviour is unchanged.
- In this section we'll look at a number of standard refactorings and see how they are used in practice:
  - Extract Method
  - Move Method
  - Replace Inheritance with Delegation
  - ...

Design Patterns

- Pattern definition:
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”
- Or in a software context:
  - A pattern as an encapsulation of an elegant design solution that can be used and reused in many different ways and contexts.
- Refactoring is often used to introduce a Design Pattern to a program.
- We'll examine a number of patterns from a standard catalogue, and do a design and implementation exercise related to patterns

‘How We Build Software’ Seminar Series

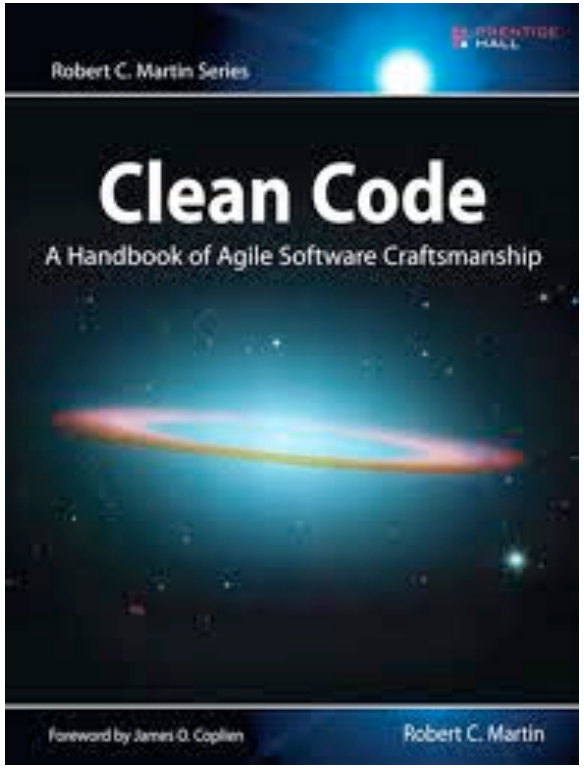
- This module incorporates a series of seminars (4-6) each entitled ‘How We Build Software’.
  - Takes place on certain Tuesdays, 3-4pm
  - Each seminar is delivered by a company involved in software development
  - Companies involved so far: Google, IBM, Realex, Openet…
- What you must do…
  - Attend each seminar
  - Ask questions on an assigned topic
  - Write a report (forms part of your lab work)
  - Possibly present your report at end of module
- …

Primary Textbook

- There is no primary recommended text.
- The following slides have details of some of the books that this module is based on.
  - In each case, only parts of the text are covered
  - None of these is a required purchase
- There is a lot of information to be found on the web in these areas as well.

"Clean Code"

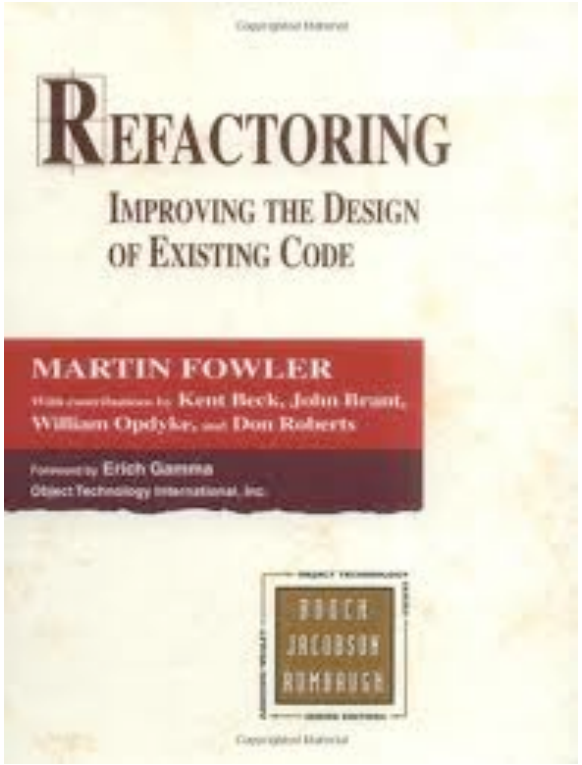
- One of the best texts that covers a lot of this module is:
  - **Robert Martin, *Clean Code*, Pearson, 2009.**
- This covers several relevant areas:
  - Testing
  - Refactoring
  - Code Smells
- This is essentially a practitioner's handbook for *software craftsmanship*.



Code Smells and Refactoring

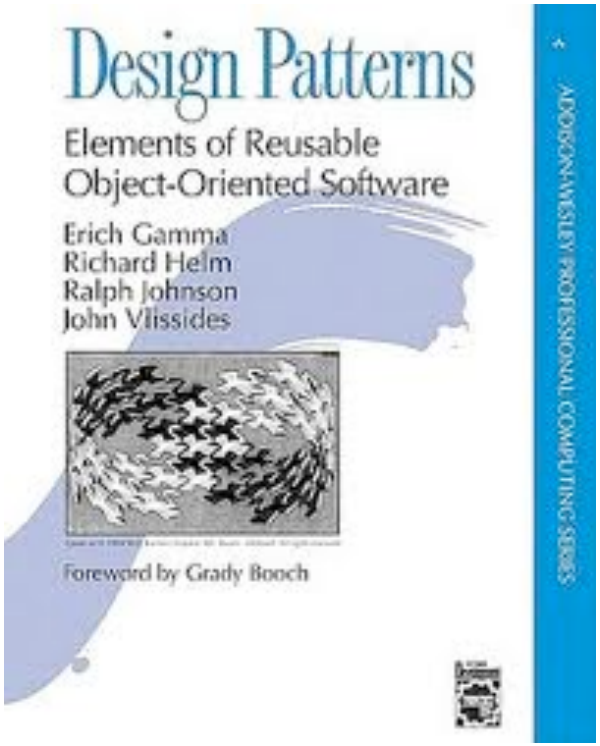
- The original text on this topic is:
  - **Martin Fowler, *Refactoring*, Addison-Wesley, 1999.**
  - UCD Library: 005.14 (1 copy)

- This book contains a catalogue of code smells and refactorings, along with advice on when to apply them and some detailed examples.



Design Patterns Text

- The best-known text in this area is:
  - **Erich Gamma *et al*, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.**
  - One of the best and best-selling books on software ever.
  - Available in Library:
    - Code: 005.12/Des
    - 3 copies in Short Loan Collection, 4 copies in General Collection.

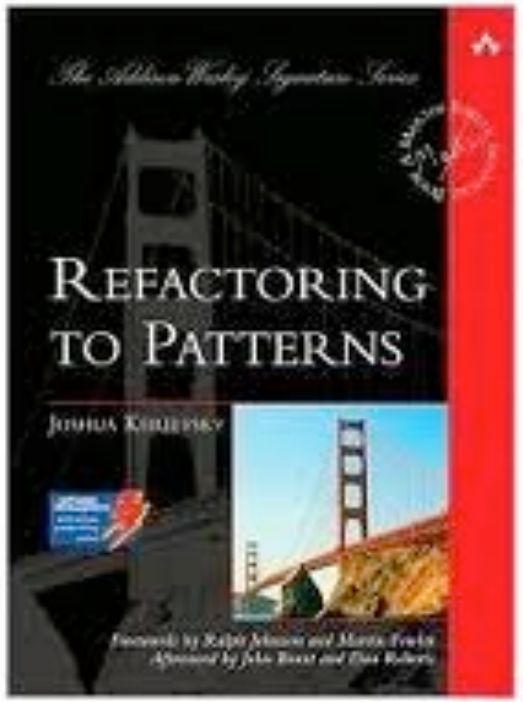


Another Design Patterns Textbook

- This is an easy-read, engaging design patterns book:
  - **Freeman et al, Head First Design Patterns, O'Reilly, 2004.**

Refactoring and Design Patterns

- See for example:
  - **Josh Kerievsky, *Refactoring to Patterns*, Pearson, 2005.**
  - UCD Library: 005.16 (1 copy)



Other Textbooks in the UCD Library

- Another UML text:
  - **Martin Fowler, *UML distilled : a brief guide to the standard object modeling language*, Addison-Wesley, 2000.**
  - Library: 005.117/FOW, 3 Gen, 3 copies in Short Loan Collection.
- Early sections on Software Quality are well worth reading in:
  - **Bertrand Meyer, *Object-Oriented Software Construction 2e*, Prentice-Hall, 1997..**
  - Library: 005.12/Mey 3 copies in Short Loan Collection.
- Easy reading on Extreme Programming:
  - **Kent Beck*, Extreme Programming Explained*, Addison-Wesley, 2000**.
  - Library: 005.1/Bec, 1 copy in General Collection.

Other Textbooks in Library

- Another patterns text:
  - **Craig Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design*, Prentice Hall 1998.**
  - Library: 005.11/LAR, 1 copy in General Collection.

- The original work on patterns:
  - **Christopher Alexander, *A Timeless Way of Building*, Oxford University Press, 1979.**
  - Library: 720.1/ALE, 1 copy in General Collection (Architecture).

Finally

- The module web page will contain the lecture slides and other useful material:
  - http://csserver.ucd.ie/~meloc/30160/ood.htm

- The next section will be an introduction to the first topic of the course, **Software Development Methodology**.