

Introduction to Object-Oriented Design

John Reekie
University of Technology, Sydney

Contributors
Lian Loke, University of Technology, Sydney

Terms of Use: Creative Commons Attribution-ShareAlike 2.5
<http://creativecommons.org/licenses/by-sa/2.5/>

“Object-oriented”

- ◆ In the beginning, there were ... object-oriented programming languages. After much argument and debate, it was decided that “object oriented” means:
 - Programs are written using classes
 - Classes have methods (or functions) and variables (or data)
 - Classes **inherit** from one another
 - Classes are instantiated to create objects
 - Objects “send messages” to each other
- ◆ Examples
 - Simula
 - Smalltalk
 - C++
 - Java

“Design”

- ◆ The creative, iterative activity associated with production of the plan enabling construction of an artifact, subject to and governed by materials, human factors, and other constraints. (My definition)
- ◆ "Design" as a verb refers to the process of originating and developing a plan for a new object (machine, building, product, etc.). ... Designing normally requires considering aesthetic, functional, and many other aspects of an object, which usually requires considerable research, thought, modeling, iterative adjustment, and re-design. (Wikipedia)

“Object-oriented design”

- ◆ The activity of designing a computer program that will be implemented in an object-oriented programming language. (Reasonable definition)
- ◆ A (futile) attempt to make everything look object-oriented. (Cynical definition)
- ◆ A design method in which a system is modeled as a collection of cooperating objects and individual objects are treated as instances of a class within a class hierarchy. (Wikipedia)

How do we do it? (Quiz)

- ◆ Start writing Java code
- ◆ Buy a set of expensive object-oriented design tools
- ◆ Write a requirements specification
- ◆ Draw a lot of diagrams
- ◆ Judiciously apply a set of tools (both thought tools and computer tools) that enable us to *model*, *understand*, and *describe* an object-oriented program (that may or may not exist yet)

Model

- ◆ A model is an abstract but “not real” description of something. The “thing” has significantly more complexity than the model, and is (more) “real.”
- ◆ In general, models enable us to analyze and predict properties of the “real” thing, in a way that is more tractable.
 - Performance models
 - Queueing models

Understand

- ◆ To be capable of using cause and effect relationships to predict the responses, properties, or behavior of something
 - If I press this button, the message will be immediately sent to the address listed there (understanding)
 - If I move the mouse like “this,” the roof will cave in (not understanding)

Describe

- ◆ To use words and diagrams to communicate understanding to someone else
- ◆ To use words and diagrams to capture, reinforce, and verify our *own* understanding
- ◆ In computing, we can consider
 - Collaboration: immediate communication, in which understanding is built up in a shared manner
 - Documentation: communication “at a distance,” in which understanding is conveyed at a later date to someone unknown.

To describe things, we need...

- ◆ Language.
- ◆ Languages have symbols (basic elements), syntax (structure), and semantics (meaning)
- ◆ There are lots of different kinds of language:
 - Natural languages (English, Chinese, French, ...)
 - Sign language
 - Formal languages (EBNF, mathematics, ...)
 - Programming languages (Java, PHP, ...)
 - Markup languages (HTML, XML, ...)
 - Visual languages (road signs, LabView, ...)

To describe object-oriented designs, we use...

- ◆ UML, the Unified Modeling Language
 - Unified: it's the result of a merging of several earlier object-oriented design languages, and an agreed-upon standard
 - Modeling: it represents models of object-oriented programs, not actual programs
 - Language: it has symbols, syntax, and semantics. It is (mostly) a visual language.
- ◆ "U" is not "Universal"
 - Not everything can be described well in UML (despite the propaganda)

UML includes...

- ◆ Notations to support analysis
 - Use case diagrams
- ◆ Notations to describe programs
 - Static structure (class) diagrams
 - Sequence diagrams
 - State diagrams
 - ..etc..
- ◆ Stuff you don't really care about
 - MOF
 - OCL
 - The pitch to buy tools

Why use UML?

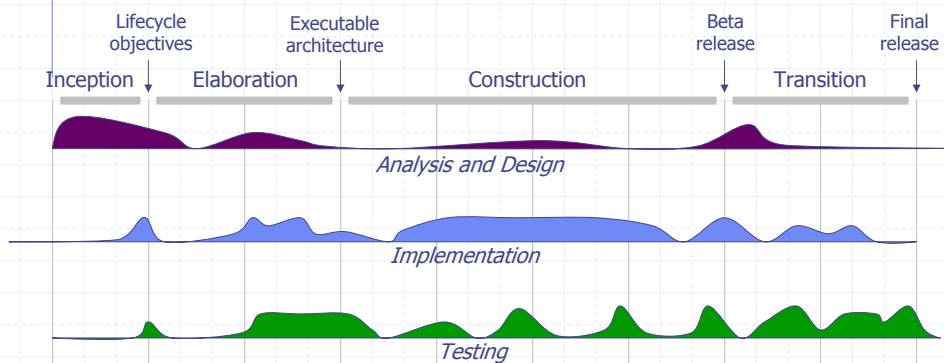
- ◆ It is a standard
- ◆ It is mature
 - Although some might say it is overgrown and stifles innovation
- ◆ There are lots of books about it
- ◆ Tools are available, even free ones
- ◆ You don't *have* to use a dedicated tool to use UML
- ◆ It has sufficient variety in notations that you can pick and choose (to some extent) those most suited to your needs. Most commonly used are:
 - Static structure (class diagrams)
 - Sequence diagrams

How do you use UML?

- ◆ OK, so here is where it gets interesting.
- ◆ Collaboration
 - Back of the napkin diagrams
 - Whiteboards
- ◆ Documentation
 - Dedicated tools
 - General-purpose tools (Visio, powerpoint, ...)
- ◆ Some software development *processes* specify UML models as artifacts that are to be produced during system development

The Unified Process

- ◆ A process framework that combines *iterative, incremental* development with a well-defined set of workflows, artifacts, roles, and activities



(This is a simplified view, for illustrative purposes)

“Iterative, incremental”

- ◆ A buzzword.
- ◆ Not just a buzzword.
- ◆ Software development proceeds with a regular, cyclic “rhythm.”
- ◆ Each cycle results in specific improvements relative to the end of the previous cycle.
- ◆ Each cycle concludes with evaluation of the work done in that cycle and re-assessment of the work to be done in the next cycle.

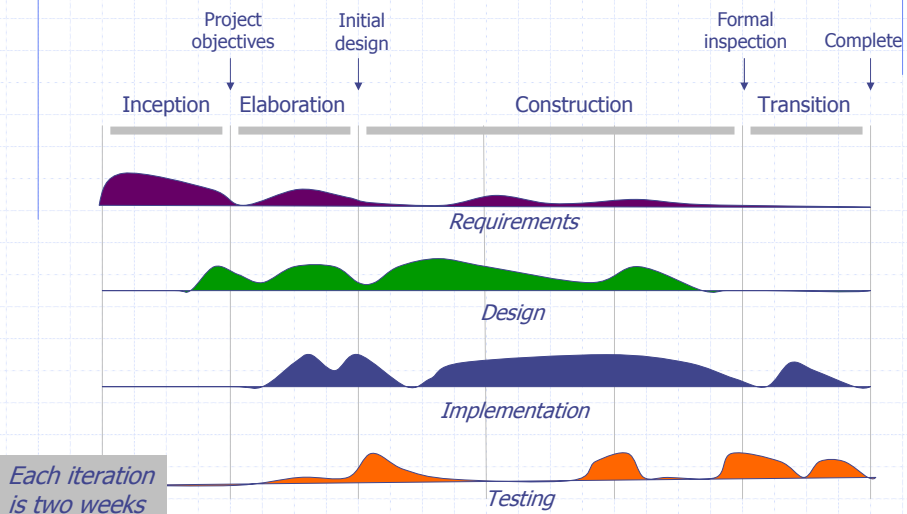
Miniature Unified Process

- ◆ For the purposes of your class project, it’s a good idea to follow an established process “in principle.” You learn something about how to develop software and your project produces a better result. (More marks is good, right?)
- ◆ But, you are not full-time software developers. And you don’t have much experience (no offence). For a class project, we need something that is very “light.”
- ◆ So: MUP (pronounced “moop”). The essence of the Unified Process. Just enough to provide some structure to your project and keep it on track.

MUP workflows and artifacts

- ◆ Requirements – use cases and user interface drawings
- ◆ Design – class diagrams and sequence diagrams
- ◆ Coding – Java in BlueJ or Eclipse
- ◆ Testing – Unit tests in Junit, system tests done by hand

Phases of the MUP



Inception

- ◆ Establish the “business case” for your system
- ◆ That means...
 1. Form into a team
 2. Decide on a project
 3. Establish ground rules (meeting times, co-ordination responsibilities etc)
 4. Agree on a schedule
 5. Identify key system elements (client, server, etc)
 6. Identify key stakeholders
 7. Identify key risks
- ◆ Deliverables
 1. Initial creation of documentation on Wiki (team)
 2. First design notes in workbook (individual)

Elaboration

- ◆ Establish the architecture and requirements
- ◆ That means...
 1. Create key use cases
 2. Use-case diagrams
 3. Sketch out (on paper) the proposed user interface
 4. Initial class diagram of all key functional elements
 5. Initial sequence diagram of important interaction
 6. Elaborate on key risks
- ◆ Deliverables
 1. Strong start to documentation on Wiki (team)
 2. Elaborated design notes in workbook (individual)

Construction

- ◆ Build and test the system. We are using three iterations.
- ◆ Over the three iterations, you should:
 1. Complete the object-oriented design
 2. Revise use cases as necessary
 3. Redesign the user interface if necessary
 4. Write the code
 5. Construct unit tests for all classes
 6. Design system tests
- ◆ Deliverables
 1. All sections of project documentation present (team)
 2. Workbook complete, except for reflections (individual)

Transition

- ◆ Wrap up and deliver the system
- ◆ That means...
 1. Fix remaining bugs
 2. Polish user interface
 3. Prepare demonstration
 4. Complete documentation on Wiki
 5. Wrap up individual workbook
 6. Prepare and deliver presentation
- ◆ Deliverables
 1. CD with all code and support files
 2. Project documentation
 3. Completed workbook

Other process models

- ◆ There are **lots**. There is no “one true way.”
- ◆ Waterfall. The “traditional” model. We must complete each task (analysis, design, etc) before commencing the next. (Generally considered now to not work very well for software development – not without modification, anyway.)
- ◆ Agile. The code is King. We know that requirements will change, so let’s not write any down at all until the moment right before we implement them. (Still controversial, although increasingly influential.)
- ◆ Staged delivery. A plan-driven model in which the software is delivered in a series of milestones. Functionality at each milestone is planned in advance, although some revision is allowed and expected. (See McConnell, *Rapid Development*.)