Object-Oriented Design

Winter Term 2003

General Information

Course title	**Object-Oriented Design**
Course number	**COMP 304B** (CRN 1255)
Prerequisites	COMP 206 (software systems). COMP 251 (data structures and algorithms), COMP 302 (programming languages and paradigms), If you do **not** have the pre-requisites for the course, the course will be **deleted** from your record by the Faculty of Science.
Course venue	Leacock 324 (check Minerva). Monday and Wednesday, 14:30 – 16:00 (starting **Monday January 6**)
Enrollment Cap	Enrollment is **limited to 65** students.
Instructor	Prof. Hans Vangheluwe McConnell Engineering room 328 tel.: +1 (514) 398 44 46 e-mail: hv@cs.mcgill.ca (send course-related mail to the course mail address)
Office hours	Monday 16:00 – 18:00
TAs	Jean-Sébastien Bolduc (jseb@cs.mcgill.ca) Marc Provost (mprovo1@cs.mcgill.ca)
TA Office hours	Wednesday 10:00 – 12:00 McConnell Engineering room 202 (Modelling, Simulation and Design Lab)
Course website	http://www.cs.mcgill.ca/∼cs304
Course e-mail	cs304@cs.mcgill.ca (start the `subject:` with `COMP 304`)

Introduction and Rationale

The complexity of software systems to be built increases with the number of components, the diversity of these components, the level of interactiveness and re-activeness, as well as with the need for integration with real-world systems. These issues are amplified when designing real-time, embedded systems. While the complexity of systems increases, so does the demand for reliability, reduced production time, and maintainability. Thanks to extensive research in algorithms and datastructures, as well as suitable –object-oriented– programming languages, re-usable libraries of "components" have been developed. Examples are the Standard Template Library (STL) for C++ and AWT/Swing for Java. Given the complexity of software-to-be-built, it is necessary to focus, not only on coding and programming, but also on the *design* of software. The design effort provides a framework for reasoning at a level of abstraction higher than that of code (which is, certainly in modern, object-oriented languages, already at a high level of abstraction compared to the level of machine instructions). Thanks to this high level of abstraction

- a higher level of complexity can be managed (*i.e.,* exceeding hundreds of thousands of lines of code);
- it is possible to focus on the essentials, abstracting away cumbersome detail;
- abstract designs are an excellent means for communicating problems and solutions;
- designs (as opposed to code-libraries) are re-usable at a level independent of implementation choices (language, libraries). The concept of design re-use has led to the proliferation of design-patterns.

Course Goals

The above introduction highlights the need for object-oriented design as opposed to coding. This will put object-oriented programming in its proper context and allow one to tackle complex problems in a structured fashion. The object-oriented paradigm, with all

the support such as object-oriented languages, type theory, tools, UML notation, and design patterns, has reached a sufficient level of maturity to allow a comprehensive presentation.

The goal of the course is to give in-depth insight in the nature of object-oriented analysis and design, integrating the following related aspects:

- understanding the (object-oriented) software development process.
- software testing as an integral part of the process;
- understanding the object-oriented paradigm;
- standardized representation for design: the Unified Modelling Language (UML);
- what are criteria for a *good* design;
- how to go from design to code in a structured (and possibly automated) fashion;
- design patterns for high-level design re-use.

Course Material

Meilir Page-Jones. *Fundamentals of Object-Oriented Design in UML.* Object Technology Series. Addison-Wesley, 2000. ISBN 0-201-69946-X.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995. ISBN 0-201-63361-2.

Several copies of these books are put on reserve at the Schulich Library.

The McGill bookstore has a reduced-price ($128.95) "package" containing both books. ISBN 0130391123.

Additional material will be made available via the course website.

Assignments and Evaluations

6 assignments (3 × 2-week, 3 × 1-week). The assignments form a series of prototypes for a spreadsheet application. The implementation language for *all assignments* is Python (http://www.python.org).

Grades distribution:

- **50%** on the implementation assignments. Weighting of individual assignments is based on difficulty and amount of work.
- **15%** on the midterm (in-class on Wednesday February 20).
- **35%** on the final exam.

A supplemental exam, worth **50%**, covers the whole course.
There is no opportunity to re-do or increase grades obtained for assignments.

Assignments will be judged on:

- correctness,
- amount of work,
- structure and completeness,
- originality,
- presentation.

Original Work

You are encouraged to help each other formulate the ideas behind assignment problems, but each student is required to submit his or her own *original* work. Handing in work that is not your own, original work as if it is your own is plagiarism. See section 15 of Student Rights and Responsibilities Handbook for more details. All re-use, collaboration, inspiration must be *explicitly* mentioned in the assignment.

Class Attendance

You are expected to attend class. Information may be made available on the web for your convenience, but is not a substitute for class attendance. Should you miss a class, you are responsible for finding out what material or announcements you may have missed.