COMP2911 Engineering Design in Computing

Semester 1, 2016

Course Outline

Contents

- Course Staff
- Course Details
- Course Aims
- Student Learning Outcomes
- Assumed Knowledge
- Teaching Rationale
- Teaching Strategies
- Assessment
- Academic Integrity and Plagiarism
- Course Schedule
- Resources for Students
- Course Evaluation and Development

Course Staff

Contact Details

Name	Role	E-Mail
Wayne Wobcke	Lecturer in charge, Lecturer Stream 1	wobcke@cse.unsw.edu.au
Brad Heap	Lecturer Stream 2 (*)	b.heap@unsw.edu.au

Note: (*) The school is highly likely to cancel Stream 2 when lecture attendance stabilizes

Course Details

Units of Credit: 6

Web Site: http://www.cse.unsw.edu.au/~cs2911/

Timetable:

	Stream 1 (*)		Stream 2 (*)	
Lecture Time	**Room**	**Lecturer**	**Room**	**Lecturer**
Tue 3.00-5.00	Rex Vowels	Wayne Wobcke	OMB 149	Brad Heap
~~Thu 12.00-1.00~~	~~Rex Vowels~~	~~Wayne Wobcke~~	~~OMB 149~~	~~Brad Heap~~

Note: (*) Thu 12.00-1.00 lecture slots will only be used when needed

Tut-Lab Time (*)	Room	Tutor/Demonstrator	E-Mail
Mon 11.00-1.00	Drum Lab	Rob Everest	robertce@cse.unsw.edu.au
Mon 5.00-7.00	Tuba Lab	Sandeepa Kannangara	s.kannangara@student.unsw.edu.au
Mon 5.00-7.00	Drum Lab	John Calvo Martinez	jcalvo@cse.unsw.edu.au
Tue 10.00-12.00	Drum Lab	Hayden Smith	hsmith@cse.unsw.edu.au
Tue 10.00-12.00	Tuba Lab	Brad Heap	b.heap@unsw.edu.au
Tue 5.00-7.00	Drum Lab	Aneita Yang	aneita.yang@unsw.edu.au
Tue 5.00-7.00	Tuba Lab	James Roberts-Thomson	jrt@cse.unsw.edu.au
Wed 12.00-2.00	Drum Lab	Sandeepa Kannangara	s.kannangara@student.unsw.edu.au
Wed 12.00-2.00	Tuba Lab	Andrew Barnett	abar654@cse.unsw.edu.au
Wed 2.00-4.00	Tuba Lab	James Roberts-Thomson	jrt@cse.unsw.edu.au
Wed 2.00-4.00	Drum Lab	Rob Everest	robertce@cse.unsw.edu.au
Thu 10.00-12.00	Drum Lab	Andrew Barnett	abar654@cse.unsw.edu.au

Thu 10.00-12.00	Tuba Lab	Brad Heap	b.heap@unsw.edu.au
Thu 1.00-3.00	Drum Lab	Rob Everest	robertce@cse.unsw.edu.au
Thu 1.00-3.00	Tuba Lab	Hayden Smith	hsmith@cse.unsw.edu.au
Thu 3.00-5.00	Tuba Lab	Aneita Yang	aneita.yang@unsw.edu.au
Thu 6.00-8.00	Drum Lab	John Calvo Martinez	jcalvo@cse.unsw.edu.au
Thu 6.00-8.00	Tuba Lab	James Roberts-Thomson	jrt@cse.unsw.edu.au

Note: (*) Tut-labs may be cancelled depending on enrolments

Course Aims

COMP2911 covers the theory and practice of object-oriented design and programming with an emphasis on design patterns and problem-solving algorithms. The course also provides an introduction to agile software processes, user interface design and programming, and concurrency in multi-threaded systems. It is a third course in programming, so students are assumed to be competent C programmers who can understand and use abstract data types and analyse program efficiency. COMP2911 is a prerequisite for some third and fourth stage Computer Science and Software Engineering courses and emphasizes professional issues relevant to working in the software industry.

There are a number of more specific themes and objectives.

- *Object-Oriented Design.* To introduce the basic principles of object-oriented design, including abstraction, modularity, polymorphism, interface design, design patterns and UML.
- *Object-Oriented Programming.* To introduce the object-oriented programming paradigm, including concepts such as objects, classes, inheritance, encapsulation, programming by contract, test driven development and unit testing.
- *Java.* To introduce the Java programming language and application development using the Eclipse IDE (Interactive Development Environment).
- *Algorithms.* To consolidate problem-solving methods introduced in previous courses in the context of object-oriented design and programming.
- *Software Processes.* To introduce and apply an agile software process such as Scrum in the planning and execution of a team-based project.
- *User Interfaces.* To introduce principles of effective user interface design and how Swing can be used to implement user interfaces in Java.
- *Concurrency.* To introduce issues in concurrency and the synchronization primitives used in multi-threaded systems in Java.

The course content aligns closely with the 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science developed by the ACM and IEEE Computer Society, as discussed in the school's most recent revision of the core curriculum. In particular, the following knowledge areas from the guidelines are covered in this course (though not all aspects of all areas are covered, and not all areas are covered to equal depth).

- *Programming Languages*
  - PL/Object-Oriented Programming
  - PL/Basic Type Systems
  - PL/Event-Driven and Reactive Programming
- *Software Engineering*
  - SE/Software Design
  - SE/Software Processes
  - SE/Software Verification and Validation
- *Information Management*
  - IM/Data Modeling
- *Intelligent Systems*
  - IS/Basic Search Strategies
- *Human-Computer Interaction*
  - HCI/User-Centered Design and Testing
- *Operating Systems*
  - OS/Concurrency

Student Learning Outcomes

The following are the intended learning outcomes of this course.

- *Object-Oriented Design*
  - Understand the principles of object-oriented design
  - Be able to follow a systematic object-oriented design process
  - Be able to interpret and use tools for object-oriented design
  - Understand and use object-oriented design patterns
- *Object-Oriented Programming and Java*
  - Understand object-oriented programming languages
  - Be able to write medium-scale object-oriented programs in Java
  - Apply systematic methods such as test driven development
  - Understand the importance of documentation and use tools to document software
  - Be able to use an IDE (Interactive Development Environment) for software development

- *Algorithms*
  - Understand and use a range of problem-solving algorithms
  - Be able to implement problem-solving algorithms in Java
- *Software Processes*
  - Understand the importance of team organization and communication
  - Be able to work within a small team in the context of a software development project
  - Be able to plan and execute a software project according a systematic software process
- *User Interfaces*
  - Become familiar with principles of effective user interface design
  - Be able to implement a user interface in Java
- *Concurrency*
  - Become familiar with basic issues in multi-threaded systems
  - Be able to use synchronization primitives in Java

This course contributes to the following UNSW graduate attributes.

- *An in-depth engagement with the relevant disciplinary knowledge in its interdisciplinary context.* The development of systems suitable for different user groups encourages consideration of the users of technology.
- *The capacity for analytical and critical thinking and for creative problem solving.* The creation of software applications requires creative approaches to design and implementation.
- *The ability to engage in independent and reflective learning.* Software design and implementation must be done independently and students benefit by reflecting on their experiences.
- *The skills required for collaborative and multidisciplinary work.* Through a small team-based project, students gain experience in working within a collaborative environment.
- *The skills of effective communication.* Through a small team-based project, students develop interpersonal communication skills and presentation and writing skills.

Assumed Knowledge

Students are assumed to have completed two programming courses: the first (typically) an introductory programming course based on a procedural language, the second (typically) a course on abstract data types and algorithms. Students should be able to write and debug medium-sized procedural programs, understand the elements of good programming style, understand and use abstract data types in program design and implementation, be familiar with a range of standard data structures and algorithms (such as lists, trees, graphs, etc.), and be able to analyse the computational complexity of algorithms. Students should also have some experience working in small groupwork activities.

Teaching Rationale

COMP2911 is an important course that bridges the introductory programming courses and further specialist third and fourth stage Computer Science and Software Engineering courses intended for professionals in the software industry.

The production of software encompasses a range of activities, not only the programming and analysis techniques covered in first stage courses, but also software design more broadly (including systems design, component design and user interface design), and teamwork (including team organization, communication skills and project planning, execution and monitoring). Accordingly, the course builds on students' programming skills to develop their knowledge of, and ability to apply, Software Engineering best practices in the production of high quality software. **Students should note carefully that "coding" is only one aspect of, and by no means the most important aspect of, the production of large-scale high quality software.**

More specifically, the course covers object-oriented design, object-oriented programming, the Java programming language and problem-solving algorithms in an integrated fashion, with introductions to agile software processes, user interface design and programming, and concurrency in multi-threaded systems. As far as possible, the approach is "problem-centred", where, given an example problem, the issues of software design and development are addressed in the context of the problem. Design is not seen as independent of programming, but as part of a systematic and integrated Software Engineering process.

Teaching Strategies

The course has 2 hours of lectures (a third hour is available if needed) and a combined 2 hour tutorial and lab (tut-lab) each week (lectures in Weeks 1-12 and tut-labs in Weeks 2-13). Tut-labs are **essential** for programming experience and individual feedback from tutors. Tut-labs are also important for guidance during the team-based project.

The course is offered in "flipped classroom" mode. The aim is to minimize ineffective in-class time in lectures, while encouraging students to learn at their own pace through resources provided online. Students are assumed to have read the online material for each week **before coming to the lecture** and be prepared to apply that material in worked examples (which form the basis of tut-labs). Lectures will **not** use up time reviewing all the online material but may focus on a few central points or clarify key issues in response to questions. To avoid distractions in lectures, a "no facebooking" policy in the first four rows of the lecture theatre will be adopted.

Lectures will be closely integrated with tut-labs. To coin a term, lectures will be more like "lectutes", combinations of lectures and tutorials. They have the flavour of lectures, in that they cover more conceptual material and give an overview of the main topics and their interconnections, however also have the flavour of tutorials, in that the material is highly contextualized through realistic problems, and grounded in Software Engineering practices.

Tut-labs are also organized by week, with the tut-lab task for a given week discussed in the lecture of that week.

However, students may proceed with tut-lab tasks at their own pace, and there is no requirement that students work in a tut-lab only on the task for that week (however, tutors are not expected to answer detailed questions relating to the tasks of future weeks). In addition, tutors are not required to repeat material already covered in lectures.

This course emphasizes best professional practices in Software Engineering. For practical work, it is **not** sufficient to merely develop "correct" software; it is important to ensure that any software is developed to a high standard of quality by following best practices. It is critical to attend classes both to keep up with the material and to ensure that proper design practices are being followed. Tutors are able to provide individual guidance on whether work is of an appropriate standard. **Note also that there is a large amount of misinformation on the Internet (particularly on forums and blogs, but even on the Oracle site), and the supplied course materials must be used as the primary references.**

Assessment

The assessment for this course consists of the following weighted components.

- Two programming assignments, each 10% (20%)
- Team-based project (20%)
- Final exam (60%)

The two programming assignments, each worth 10%, are marked by machine, with a portion of the marks given for design quality and programming style. The team-based project consists of an individual component (5%) and a team component (15%) making up 20%. The individual mark is based on a reflective diary shown weekly to the tutor; the team-based assessment includes a presentation/demonstration of the final system (held in Week 13). Each member of the team generally receives the same mark for the team-based component of the project. The final exam is a 2 hour written examination worth 60% covering the major aspects of the course, particularly object-oriented design and object-oriented programming using Java.

The final mark for the course is determined by adding together these component marks according to the above weighting to give a result out of 100, which is subject to further scaling.

Late submission policy for assignments: Assignments submitted late are subject to the penalty that the mark obtainable reduces by 20% of the value of the assignment per day or part-day late, up to a maximum of 3 calendar days, after which a mark of 0 is received.

Academic Integrity and Plagiarism

UNSW has instituted severe penalties for academic plagiarism. Therefore it is important for students to understand what is acceptable and unacceptable in the presentation of work for assessment.

You should **carefully** read the UNSW policy on academic integrity and plagiarism. Note, in particular, that *collusion* (working together on an assignment, or sharing parts of assignment solutions) is a form of plagiarism.

In essence, as applied to COMP2911, copying or sharing program code for individual assignments counts as plagiarism and is unacceptable. In the team-based programming project, sharing code *amongst team members* is acceptable, but sharing code between (members of) different teams is unacceptable. Copying code from the Internet is unacceptable in any circumstances. Assignment submissions are checked for similarity, both with one other and with code on the Internet. The school maintains a register of students with confirmed plagiarism offences. The penalties for plagiarism range from receiving 0 or negative marks for an assignment, through receiving a mark of 00 FL for the course, to expulsion from UNSW (for repeat offenders). **Students should be aware that the school takes plagiarism very seriously, and that these penalties are applied routinely in COMP2911.** Note that allowing someone else to copy your work counts as plagiarism, and makes you liable to a penalty, even if you can prove that the work was yours originally. **Students are encouraged to use a version control system to regularly back up work, however it is essential to ensure that all repositories (such as on github) are private, to prevent the possibility of plagiarism.**

Net etiquette: There are many online Java programming forums. Do **not** use these forums to post questions that essentially ask for solutions to the assignments. These forums are for professional Java programmers, and generally the user community will discourage use of a forum in this way. Use such forums only to ask arcane Java questions, and only then after consulting the Java API documentation.

Course Schedule

The following is the rough sequence of topics by week. More details will be provided during the semester.

Week	Topic	Notes
1	Overview	
2	Object-Oriented Programming	
3	Programming by Contract	
4	Object-Oriented Design	
5	Generic Types and Polymorphism	Assignment 1 due

6	Basic Search Algorithms	
7	Problem-Solving Algorithms	
8	Agile Software Processes	
9	User Interface Design	Assignment 2 due
10	Design Patterns	
11	Concurrency	
12	Review	Project due
13	Project Assessment	

Resources for Students

Supplementary Material

Much of the material on object-oriented design and programming for this course will be provided as web references **for reading prior to the lectures**.

The following additional resources may serve to expand on the ideas developed in this course.

> Horstmann, C. *Object-Oriented Design and Patterns.* Second Edition. John Wiley and Sons, Hoboken, NJ, 2006.
> *Textbook on object-oriented design and programming, emphasizing design patterns.*
>
> Liskov, B. & Guttag, J. *Program Development in Java*. Addison-Wesley, Reading, MA, 2001.
> *Discusses different notions of program abstraction and their use in Java programming.*
>
> Gamma, E., Helm, R., Johnson, R. & Vlissides, J. *Design Patterns*. Addison-Wesley, Reading, MA, 1995.
> *The basic reference on design patterns that provides many more examples than given in the course.*

Java Resources

This course involves Java programming. There is no need to buy a Java reference, however the online Java Tutorials may be helpful and using the online Java API documentation is essential. If you do use a Java reference, make sure that it is for Java Standard Edition Version 5 or above. Java SE 7 is installed in the labs so that is preferred. Beware that Java SE 8 includes some small but significant changes to Java SE 7. **It is your responsibility to ensure that your assignments run on lab machines before submission.**

> Java Tutorials
> Java SE 7 APIs
> Java SE 8 APIs

The course also involves Java development using Eclipse, an IDE (Interactive Development Environment). The following links are to useful Eclipse resources. The version of Eclipse installed in the labs is Eclipse IDE for Java EE Developers (Mars), which is listed at the top of the package solutions on the Eclipse Downloads page. Eclipse IDE for Java Developers (Mars) can also be used. Note that Eclipse Mars requires at least Java SE 7, and automatically includes support for Java SE 8.

> Eclipse Downloads
> Eclipse Mars Help

Course Evaluation and Development

CSE courses are evaluated by student survey each time they are taught. The survey includes standard questions asked of all comparable courses so that it is possible to compare COMP2911 with other relevant UNSW courses, and also includes space for free-form comments. Survey responses are anonymous. The completed survey forms are analysed statistically by someone independent of the course staff, and the results, including free-form comments, are made available to the lecturer in charge *after* grades have been reported and released.

Feedback from previous offerings of COMP2911 has indicated that the workload has been considered heavy compared to other CSE courses. Since the course involves learning a new paradigm (object-oriented design and programming) which is typically learnt by practical experience, and also includes a project involving programming in teams, this course does require an extensive time commitment. However, in line with this feedback, the number of programming assignments was reduced to two, with one small team-based project.

This course was thoroughly revised in response to feedback from student representatives in 2014. First, the textbook (which was highly regarded when it was introduced) was dropped, as stureps strongly indicated that they preferred course content to be "original". Second, as an alternative to the traditional lecture format, the course was offered in a "flipped classroom" mode, with background material for each lecture expected to be read by students **in advance**. Third, the lecture sequence was reorganized to align more closely with tut-labs, also with a 2 hour tut-lab replacing the former 1 hour tutorial and 2 hour lab. The aim was to focus on key concepts in both lectures and tut-labs. Fourth, the exam, which was previously a hurdle to passing the course, was no longer a hurdle. Tut-lab participation became optional, though strongly advised.

The course experience in 2015 was markedly improved. Lecture attendance was higher throughout the semester and students learnt more as a result. Tutors could focus on helping students rather than assessing lab tasks. Assignment marks were higher even though the first assignment was more difficult, software projects were better managed, and final exam marks were higher. The course revisions were proven to be highly successful, and the improvements are retained for this year.

Wayne Wobcke
Last updated: 19 November 2021
CRICOS Provider No. 00098G