COMP2511 Object-Oriented Design and Programming

Semester 1, 2018

Course Outline

Contents

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

Course Staff

Contact Details

Name	Role	E-Mail
Wayne Wobcke	Lecturer in charge, Lecturer Stream A	w.wobcke@unsw.edu.au
Aarthi Natarajan	Lecturer Stream B	a.natarajan@unsw.edu.au

Course Details

Units of Credit: 6

Web Site: http://www.cse.unsw.edu.au/~cs2511/

Timetable:

	Stream A		Stream B	
Lecture Time	**Room**	**Lecturer**	**Room**	**Lecturer**
Wed 11.00-1.00	Rex Vowels	Wayne Wobcke	ChemSc M17	Aarthi Natarajan

Tut-Lab Time (*)	Room	Tutor/Demonstrator	E-Mail
Mon 4.00-6.00	Bongo Lab	John Calvo Martinez	j.calvomartinez@unsw.edu.au
Mon 6.00-8.00	Bongo Lab	Andrew Vo	andrew.vo@student.unsw.edu.au
Tue 9.00-11.00	Bongo Lab	Angus Fletcher	a.fletcher@unsw.edu.au
Tue 9.00-11.00	Tabla Lab	Stephanie Chua	stephanie.chua@unsw.edu.au
Tue 3.00-5.00	Drum Lab	Angus Fletcher	a.fletcher@unsw.edu.au
Wed 9.00-11.00	Tabla Lab	Andrew Vo	andrew.vo@student.unsw.edu.au
Wed 9.00-11.00	Bongo Lab	Brittany Evat	b.evat@unsw.edu.au
Wed 2.00-4.00	Bongo Lab	Sandeepa Kannangara	s.kannangara@unsw.edu.au
Wed 2.00-4.00	Tabla Lab	Rob Everest	r.clifton-everest@unsw.edu.au
Thu 9.00-11.00	Bongo Lab	Sandeepa Kannangara	s.kannangara@unsw.edu.au
Thu 11.00-1.00	Tabla Lab	Stephanie Chua	stephanie.chua@unsw.edu.au
Thu 11.00-1.00	Bongo Lab	Gary Bai	g.bai@student.unsw.edu.au
Fri 9.00-11.00	Bongo Lab	Gary Bai	g.bai@student.unsw.edu.au
Fri 9.00-11.00	Tabla Lab	Dillon Giacoppo	dillon.giacoppo@unsw.edu.au
Fri 9.00-11.00	Drum Lab	Rob Everest	r.clifton-everest@unsw.edu.au
Fri 11.00-1.00	Drum Lab	Brittany Evat	b.evat@unsw.edu.au

Fri 11.00-1.00	Tabla Lab	John Calvo Martinez	j.calvomartinez@unsw.edu.au
Fri 1.00-3.00	Tabla Lab	Dillon Giacoppo	dillon.giacoppo@unsw.edu.au

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

Course Aims

COMP2511 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. COMP2511 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 project.
- *User Interfaces.* To introduce principles of effective user interface design and how Swing or JavaFX 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 project, students gain experience in working within a collaborative environment.
- *The skills of effective communication.* Through a small team 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. **An exceptional situation arises in 2018, where around half the cohort have completed COMP1531 Software Engineering Fundamentals (a new prerequisite for COMP2511 introduced during the curriculum review), and have some experience with object orientation. However, this knowledge cannot be assumed because a large number of students have not completed COMP1531.**

Teaching Rationale

COMP2511 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.

The course organization is inspired by agile principles. First, in restructuring this course, the main idea of lean management, the elimination of waste, led to the removal of all course content that did not contribute *directly* to the learning outcomes, streamlining the weekly lecture/tut-lab series. Second, following agile software development principles, working code is paramount (not just any "working" code, but working code developed through sound software design processes). The course is not organized "top down" as a series of high-level topics (in contrast to the Horstmann textbook), rather the course starts with basic object-oriented programming concepts and techniques (working code is produced in Week 1), each week augmenting the knowledge built up so far to enable more complex designs and systems, culminating in the team project which incorporates aspects of the object-oriented design process, design patterns, algorithms, agile methods and user interface design and programming.

Teaching Strategies

The course has 2 hours of lectures and a combined 2 hour tutorial and lab (tut-lab) each week (lectures in Weeks 1-12 and tut-labs in Weeks 2-13). Lectures are **essential** for providing conceptual foundations for the course and preparation and understanding of tut-lab exercises. Tut-labs are **essential** for programming experience and individual feedback from tutors. Tut-labs are also important for guidance during the team project.

The course is offered in "blended learning" 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 studied the online material for each week **before the lecture** and be prepared to apply that material in worked

examples (which form the basis of tut-labs), then review the material after the lecture and/or tut-lab. To aid this learning process, a *resources guide* for each week will outline which aspects of the resources to focus on for preparation and review. 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.

Lectures will be closely integrated with tut-labs. Lectures will be more like a combination of traditional 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. A perennial issue is that therefore, some students will attend the tut-lab before the lecture, and some after. However, students may proceed with tut-lab exercises 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, and do not have the time to give all but a brief summary of the main topics of the lecture.

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. Moreover, work builds on that of previous weeks, so students need to ensure they continue to apply knowledge from one week to the next. 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 – in the case of programming by contract), and the supplied course materials must be used as the primary references. Be particularly wary of the unofficial Facebook group for the course, which will not be used by tutors, and includes students who may have studied a different version of the course in past years.**

Assessment

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

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

The two programming assignments, each worth 10%, are both auto-marked and marked manually for design quality and programming style. For correct auto-marking, it is the student's responsibility to ensure that the submitted code runs on the school environment using the version of Java installed in the labs. The team project mark comprises an individual component (5%) and a team component (15%) making up 20%. The individual component mark assesses contributions to software development in the team project using as evidence a reflective diary shown weekly to the tutor and peer evaluations entered in Moodle using the Team Evaluation tool; the team component mark is based on a presentation/demonstration of the final system (held in Week 13). Each member of a team *generally* receives the same mark for the team component of the project, however, where individual contributions to software development are highly unequal, individual marks for this component will be adjusted to reflect the level of contribution. The final exam is a 2 hour written examination worth 60% covering the major aspects of the course, particularly object-oriented design, design patterns 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 1 and 2, late submissions 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. There is no provision for a late penalty for the project: late project submissions receive a 0 mark, or a mark at the discretion of the lecturer in charge.

Student Conduct

The **Student Code of Conduct** (Information, Policy) sets out what the University expects from students as members of the UNSW community. As well as the learning, teaching and research environment, the University aims to provide an environment that enables students to achieve their full potential and to provide an experience consistent with the University's values and guiding principles. A condition of enrolment is that students *inform themselves* of the University's rules and policies affecting them, and conduct themselves accordingly.

In particular, students have the responsibility to observe standards of equity and respect in dealing with every member of the University community. This applies to all activities on UNSW premises and all external activities related to study and research. This includes behaviour in person as well as behaviour on social media, for example Facebook groups set up for the purpose of discussing UNSW courses or course work. Behaviour that is considered in breach of the Student Code Policy as discriminatory, sexually inappropriate, bullying, harassing, invading another's privacy or causing any person to fear for their personal safety is serious misconduct and can lead to severe penalties, including suspension or exclusion from UNSW.

If you have any concerns, you may raise them with your lecturer, or approach the School Ethics Officer, Grievance Officer, or one of the student representatives.

Plagiarism

UNSW has an ongoing commitment to fostering a culture of learning informed by academic integrity. All UNSW staff and students have a responsibility to adhere to this principle of academic integrity. Plagiarism undermines academic integrity and is not tolerated at UNSW. Plagiarism is defined as "using the words or ideas of others and presenting them as your own" but also covers "students providing their work to another student". UNSW and CSE treat plagiarism as academic misconduct, which means that it carries penalties as severe as being excluded from further study at UNSW.

There are several online resources to help you understand what plagiarism is and how it is dealt with at UNSW:

> Plagiarism and Academic Integrity
> UNSW Plagiarism Procedure

Make sure that you read and understand these. Ignorance is not accepted as an excuse for plagiarism. In particular, you are also responsible for your assignment files being inaccessible by anyone but you by setting the correct permissions in your CSE directory and code repository. Note also that plagiarism includes paying or asking another person to do a piece of work for you and then submitting it as your own work. *Copying* (taking ideas and/or text from other students or the Internet and presenting them as your own) and *collusion* (working together on an assignment, or sharing parts of assignment solutions) are forms of plagiarism. In COMP2511, this applies particularly to the programming assignments, where all submitted code must be your own work.

Take the time to read the full UNSW policy on academic honesty and plagiarism. The pages below describe the policies and procedures in more detail:

> Student Code Policy
> Student Misconduct Procedure
> Plagiarism Policy Statement
> Plagiarism Procedure

You should also read the following page, which describes your rights and responsibilities in the CSE context:

> Essential Advice for CSE Students

In essence, as applied to COMP2511, copying or sharing program code for individual assignments counts as plagiarism and is unacceptable. In the team 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 COMP2511. There were 23 cases of plagiarism in COMP2911 prosecuted in 2017.** 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. This applies even after you have completed the course.**

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	Introduction	
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

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** and **review after the lectures and tut-labs**. There will be a *resources guide* for each week. Copies of lecture slides will be provided for each lecture, which give a rough outline of the structure and main points of the lecture, but are not comprehensive notes. Students are advised to attend lectures in person and view the lecture recordings (see below).

Textbook

While accurate web references have been provided, some of the sources are more advanced than needed for this course and, as authors use slightly different notations from one another, taken as a whole the sources suffer from slight inconsistencies. Therefore references to the old textbook for this course are also provided for each lecture. The book is currently out of print, but electronic copies are available online.

> Horstmann, C. *Object-Oriented Design and Patterns.* Second Edition. John Wiley and Sons, Hoboken, NJ, 2006.

The textbook has the benefits of a very clear presentation style, excellent discussion of the central issues, exemplary Java coding style, and consistent use of notation. However, the book is written presuming a knowledge of Java, and many of the examples are needlessly complicated. Some fundamental object-oriented programming concepts are introduced all at once and quite late in the book, causing a mismatch between the order of material in the book and in the lectures and labs. In short, this book is recommended as an aid to understanding the basic principles of the course (important for the exam) and as a series of examples demonstrating Java programming style (useful for lab exercises and assignments).

Supplementary Material

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

> 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.*

Lecture Recordings

Based on the trial of the lecture recording system in 2017, lecture recordings will again be distributed. However, due to the general deficiencies of lecture recordings outlined below, recordings can **not** be recommended as an adequate substitute for attendance, so students are encouraged to attend lectures in person. In any case, it is very important to keep up to date in this course, so even if relying only on recordings, students should make sure to view each lecture as soon as possible and preferably before the associated tut-lab.

There are several deficiencies with lecture recordings that mean recordings cannot capture the full "experience" of lecture attendance, even if the system is working correctly.

- The output of only one projector (the primary projector) is recorded
- Audio and video quality may not be satisfactory at all times
- Use of the whiteboard/blackboard is not recorded
- Copyright protected material is not always able to be recorded (see below)
- Questions from students and class discussions may not be able to be recorded (see below)

In partial mitigation of these issues, the following practices will be adopted.

- The most relevant material will be displayed on the primary projector
- The lapel microphone will be used to ensure adequate audio quality
- An iPad or document camera will be used for sketches and diagrams
- Lecture recordings may be stopped while copyright protected material is presented
- Lecture recordings may be stopped during class discussions

The following elaborates on the issues concerning copyright and privacy. First, the copyright exceptions for educational institutions explain what is permissible during a course of research or study for lecturers and students. Note that images are covered by the Statutory Text and Artistic Licence defined in Part VB of the Copyright Act, which allows them to be recorded when presented in a lecture, but web pages, computer software and videos are not, so cannot be recorded if this is not permitted by the copyright holder (e.g. the YouTube standard licence does not allow recordings). **Note that students do not have permission to make a copy of a lecture recording, or to redistribute (send to anyone or post on the Internet) lecture recordings. In addition, students are not permitted to make their own copies (photographs or recordings) of any part of a lecture or tut-lab.**

Second, the UNSW privacy policy prohibits disclosure, without consent, of personal information that may be used to identify an individual. This includes voice recordings of students that are captured by the lecture recording system. A notice warning that lectures could be recorded is displayed at the door to any lecture theatre where the recording system is enabled, but the issue of student consent is not mentioned. However, based on the trial of the system in 2017, it is

extremely unlikely that the lecture recording system captures sufficient information to enable the identity of a speaker in the audience to be ascertained. Hence lecture recordings will *not* be stopped during class discussions. On the other hand, before or after the lecture or during breaks, when students are speaking to the lecturer close to the console, the recording system will be stopped, as in these situations, the system does record students' voices quite clearly. **If any student has any reservation about being recorded, they should notify the lecturer whenever this concern arises, and lecture recordings will be stopped while the student is speaking.**

Java Resources

This course involves Java programming. The Java Tutorials are helpful for learning the specifics of Java, and the online Java API documentation is essential for use on a day-to-day basis. It is important to be aware of the differences in the various versions of Java. The version installed in the labs is OpenJDK 8 (an open source implementation of Java SE 8), so Java SE 8 is preferred for use on a personal computer (be sure to download the latest version of the JDK, not the JRE). Java SE 9 includes small changes to Java SE 8, and OpenJDK 8 is essentially the same as Oracle's implementation of Java SE 8, but differs in the JavaFX libraries. **It is your responsibility to ensure that your (auto-marked) assignments run on lab machines, i.e. under OpenJDK 8 in the lab Linux environment, before submission.** For the project, there is some flexibility which version of Java (Oracle Java SE 8 or OpenJDK 8) is used, but the main requirement is not to use additional (non-standard) third party libraries.

> Java Tutorials
> Java SE 8 API

The course also involves Java development using Eclipse, an IDE (Interactive Development Environment). The version of Eclipse installed in the labs is Eclipse Oxygen.2 (Version 4.7.2).

> Eclipse Downloads
> Eclipse Oxygen Help

The course also builds on the use of agile methods in COMP1531 Software Engineering Fundamentals. Mountain Goat Software provides useful and accurate descriptions of the use of agile methods in Software Engineering.

> Mountain Goat Software

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.

From 2018, COMP2511 Object-Oriented Design and Programming replaces COMP2911 Engineering Design in Computing, and is essentially the same course. In anticipation of the COMP2511 course description eventually approved by the school, and in response to feedback from student representatives, COMP2911 was thoroughly revised in 2015. First, the textbook that had been introduced to address the previous lack of student resources, and which was well received, 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 "blended learning" mode, with background material for each lecture expected to be read by students in advance, and lectures focusing on key concepts. Third, the lab exercises were completely revised, with a 2 hour tut-lab replacing the former 1 hour tutorial and 2 hour lab, reducing excessive contact hours. Fourth, the lecture sequence was reorganized to better inregrate with the new tut-labs. Fifth, the assessment was changed so that the exam, which was previously a hurdle to passing the course, was no longer a hurdle. Sixth, the length of the exam was reduced. Seventh, tut-lab participation became optional, though strongly advised.

The course experience since then has markedly improved. Lecture attendance was higher, lectures were more interactive 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 exam marks were significantly higher. The course revisions were highly successful.

In 2017, the main changes were that Stream B lectures ran for the whole semester and lecture recordings were released for both streams. Other improvements included additional resources: summary slides of each lecture plus supplementary slides for the Stream B lectures, and references to the old textbook as an aid to understanding key concepts and as an exemplar of good programming style. Finally, the exam was reduced even further.

In 2018, there will be three main changes. First, the *class forum* will be reintroduced (the forum was abandoned several years ago after students did not use it at all one year). This will address the problem of misinformation circulating on the unofficial Facebook group for the course, and improve the consistency of information between the lecture streams. Second, there will be a new *resources guide* produced for each week to address the issue that, as the number of resources has increased, it has become increasingly difficult to use them to best effect. This should also help make the objectives for each lecture more clear. A third change is that, due to the COMP1531 prerequisite, there will be a more rigorous approach to *agile methods* in the team project, and, based on a successful trial last year, the Moodle Team Evaluation tool will be used for assessing individual contributions to the project.

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