COMS3009A: Software Design

Course Outline

Semester 1, 2023

1 Lecturer

Name:	Mr. Tamlin Love
Email:	tamlin.love@wits.ac.za
Office:	UG11
Consultation Hours:	Monday 12:00 - 14:00

2 The Course

2.1 Welcome

The Computer Science curriculum includes a software design component that is aligned with the recommendations prescribed by the ACM and IEEE organisations[1]. This alignment is different from what would be covered in a typical software engineering degree as it is more focused on those aspects of software engineering deemed essential to a computer scientist.

Specifically the students should already have a strong understanding of the design and implementation of algorithms from previous courses. This course will cover higher level design principles, focusing on object oriented analysis and design, functional decomposition, event driven design, aspect oriented and service oriented architectures. Students will also be exposed to design patterns, and test driven development.

The students will learn the above content through a combination of lectures and the completion of a large software project. The large software project will go from a project proposal/statement of work to user stories and all the way through to a completed product. The Scrum methodology will be used to achieve this.

2.2 Communication Channels

- In-Person Lectures and Labs
- Moodle (Content related work)
  - Moodle Announcements
  - Software Design Overflow (Moodle Overflow)
- Google Meet (Schedule via `https://calendly.com/tamlinlovewits/consultation`, email me if you are unable to make this time)
- Email

[1]http://www.acm.org/education/curricula-recommendations

2.3 Topics Covered

Software Process

- Introduction to software process models (e.g., waterfall, incremental, agile)
- Programming in the large vs. individual programming
- Evaluation of software process models

Tools and Environments

- Software configuration management and version control
- Release management
- Requirements analysis and design modelling tools
- Testing tools including static and dynamic analysis tools
- Programming environments that automate parts of program construction processes (e.g., automated builds):
  - Continuous integration
- Tool integration concepts and mechanisms

Requirements Engineering

- Describing functional requirements using, for example, use cases or user stories
- Properties of requirements including consistency, validity, completeness, and feasibility
- Non-functional requirements and their relationship to software quality

Software Design

- System design principles:
  - levels of abstraction (architectural design and detailed design),
  - separation of concerns,
  - information hiding,
  - coupling and cohesion,
  - reuse of standard structures
- Object-oriented analysis and design
- Design patterns
- Software architecture concepts and standard architectures (e.g. client-server, tiered (n-tiered), pipes-and-filters)
- Internal design qualities, and models for them:
  - efficiency and performance,
  - redundancy and fault tolerance,
  - traceability of requirements
- External design qualities, and models for them:

    - functionality,
    - reliability,
    - performance and efficiency,
    - usability,
    - maintainability,
    - portability

Software Verification and Validation[2]

- Testing fundamentals:
    - Unit, integration, validation, and system testing
- Test-driven development

2.4 Expectations

It is useful if you have knowledge or are in a position to acquire knowledge of some of the following:

- The Object Model and Object Oriented Programming
- Frameworks such as Flutter, React, Node or Django

3 Outcomes

Having successfully completed this course, the student will be able to:

- Describe the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile). [Familiarity]
- Differentiate among the phases of software development. [Familiarity]
- Explain the concept of a software life-cycle and provide an example, illustrating its phases including the deliverables that are produced. [Familiarity]
- List the key components of a use case or similar description of some behaviour that is required for a system. [Familiarity]
- Identify both functional and non-functional requirements in a given requirements specification for a software system. [Usage]
- Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation. [Familiarity]
- Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design. [Usage]
- Construct models of the design of a simple software system that are appropriate for the paradigm used to design it. [Usage]
- Within the context of a single design paradigm, describe one or more design patterns that could be applicable to the design of a simple software system. [Familiarity]

[2]Elective

- Identify the fundamental principles of test-driven development methods and explain the role of automated testing in these methods. [Familiarity]
- Conduct an inspection or review of software source code for a small or medium sized software project. [Usage]
- Explain the relationships between the requirements for a software product and its design, using appropriate models. [Assessment]
- Investigate the impact of software architectures selection on the design of a simple system. [Assessment]
- Apply simple examples of patterns in a software design. [Usage]
- Select suitable components for use in the design of a software product. [Usage]
- Explain how suitable components might need to be adapted for use in the design of a software product. [Familiarity]

4 Teaching Methods and Course Structure

A blended learning approach that makes use of the following teaching/learning methodology, opportunities and experiences is used:

- lectures;
- lab sessions;
- subject-related learning materials; and
- consultations with the course lecturer/tutors.

The following scheduled learning opportunities will take place. The content of all labs and lectures may be used for assessment opportunities.

Format	**When**	**Time**	**Location**
Consultation Period	Mondays	12:00-14:00	Online
Labs	Mondays	14:15-17:00	MSL
Lectures	Wednesdays	8:00-9:45	MPT (DJ Du Plessis)

1. **Content**
    - Every week, lectures will take place during the scheduled lecture times.
    - You are required to attend every lecture. You are welcome to schedule a meeting with the lecturer/tutor for consultation on the course content.
    - You are required to attend all assessments (lab assessments, class tests, project scrum meetings and the exam).
2. **Project**
    - You will be divided into groups of 4-6 people.
    - Each group will be given the chance to select 3 projects from the projects list provided. Your choices should include 3 different projects in descending order of preference (most to least favourite).
    - A project will be assigned to you based on your selections. You are then required to complete the projects using the **scrum** methodology.

4.1 Attendance

Attendance of all lectures and assessments is required. Registration may be taken. Participation can count towards your class assignment mark.

4.2 Course Schedule

Learners are requested to review the work schedule below, paying particular attention to the dates of the assessment opportunities. Please note that the schedule pertaining to the material covered is tentative and subject to change.

Date	**Format**	**Content**
22 Feb	Lecture	Lecture 1: Introduction & Software Configuration Management
1 Mar	Lecture	Lecture 2: Requirements
3 Mar	Project	Selection of Projects
6 Mar	Project	Sprint 1 Starts
8 Mar	Lecture	Lecture 3: Testing
13 Mar	Lab	Lab Assessment 1
15 Mar	Lecture	Lecture 4: The Object Model
22 Mar	Lecture	Lecture 5: Software Architecture
29 Mar	Lecture	Lecture 6: Architecture Styles Pt. 1
3 Apr	Test	Class Test 1
3 Apr	Project	Sprint 2 Starts
5 Apr	Lecture	Lecture 7: Architecture Styles Pt. 2
Mid-Term Break		
17 Apr	Lab	Lab Assessment 2
19 Apr	Lecture	Lecture 8: Object-Oriented Design
24 Apr	Project	Sprint 3 Starts
26 Apr	Lecture	Lecture 9: Design Patterns Pt. 1
3 May	Lecture	Lecture 10: Design Patterns Pt. 2
8 May	Lab	Lab Assessment 3
10 May	Lecture	Lecture 11: Design Patterns Pt. 3
15 May	Project	Sprint 4 Starts
17 May	Lecture	Lecture 12: Catch-Up
22 May	Test	Class Test 2
24 May	Lecture	Lecture 13: Catch-Up
29 May	Lab	Lab Assessment 4
31 May	Lecture	Lecture 14: Catch-Up
Exam Period		

5 Assessments

An integrated approach to assessment whereby assessment forms an integral part of teaching and learning. The assessment approaches used in this course are as follows:

Formative Assessments The learner is assessed throughout the semester in the form of two class tests, four lab assessments and a group project.

Summative Assessments The learner is required to complete a written examination that is representative of all the work covered at the end of the semester.

5.1 Project

You will get together in groups of 4-6 people (larger or smaller groups are not allowed) and select a project from the provided list. Over the course of the semester, you will work closely with a client to produce their desired software, following the scrum methodology. At the end of each development sprint, your progress will be assessed.

By the end of the semester, your software should be functional and user friendly, to the satisfaction of the client. The client's impression of the software may form part of the project mark.

5.2 Lab Assessments

Four lab assessments will assess your ability to implement important aspects of the software development process, as covered by the course content and project. Students who engage with every aspect of the project should not have any trouble with these assessments. Each assessment will be out of 1 mark, and the average of these lab assessments will be multiplied by your final project mark to determine your lab mark.

For example, if you receive 75% for your project, and your lab assessment marks were 1, 1, 0.5 and 1 (average: 0.875), then you would receive 75% for your project (weighted 15%) and 66% for your lab mark (weighted 15%).

5.3 Mark Breakdown

Class Tests	**30%**
Project	**15%**
Lab Mark (Project Mark $\times$ Lab Assessments)	**15%**
Exam	**40%**

5.4 Academic Integrity

Refer to the computer science general undergraduate course outline's statement on academic integrity.

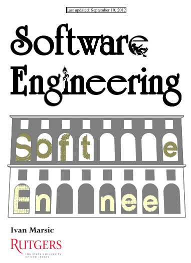
6 Course Material

Students are encouraged to make use of the available resources for the module. Sources include: lecture notes and other information posted on Moodle.

You are encouraged to use Google, YouTube, OpenCourseWare, StackOverflow and any other online resources.

6.1 Textbook

The following textbooks (available free online) may be helpful.

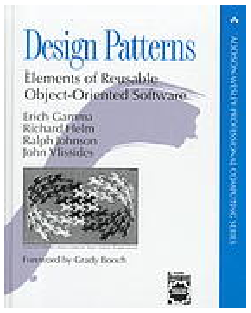


Software Engineering

Ivan Marsic

RUTGERS

Title:	Software Engineering
Edition:	September 2012
Author:	Ivan Marsic
Publisher:	Rutgers University
ISBN:	NA
Web Site:	`http://www.ece.rutgers.edu/~marsic/books/SE/`



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Title:	Design Patterns: Elements of Reusable Object-Oriented Software
Edition:	1994
Author:	The "Gang of Four"
Publisher:	Addison-Wesley
ISBN:	0-201-63361-2
Web Site:	`http://www.cs.unc.edu/~stotts/GOF/hires/contfso.htm`

6.2 Reading List

Pro Git: `https://git-scm.com/book/en/v2`

How to Write a Software Project Proposal: `http://www.ece.rutgers.edu/~marsic/Teaching/SE/proposal.html`

iso-architecture.org: `http://www.iso-architecture.org/`

Manifesto for Agile Software Development: `http://agilemanifesto.org/`

Microservices vs. Service-Oriented Architecture: `http://www.oreilly.com/programming/free/microservices-vs-service-oriented-architecture.csp`

Patterns and AntiPatterns: `https://sourcemaking.com/`

Scrum Card: `http://scrumreferencecard.com/scrum-reference-card/`

Software Architecture: `http://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm`

6.3 Viewing and Listening List

Scrum `http://scrumtrainingseries.com/`

YouTube `https://www.youtube.com/watch?v=D8vT7G0WATM&list=PLF6BFA8BAEDF6CE70`

6.4 Software

Some UML software options include:

umbrello `https://umbrello.kde.org/`

modellio `http://www.modelio.org/`

uml designer `http://www.umldesigner.org/overview/`

papyrus `http://www.eclipse.org/papyrus/index.php`

Some source control and test driven development options:

Travis CI: `https://travis-ci.org/`

Coveralls: `https://coveralls.io/`

GitHub: `https://github.com/`

TAIGA `https://taiga.io/`

Some cloud infrastructure:

IBM LinuxOne: `https://developer.ibm.com/linuxone/`

OrangeScrum: `https://www.orangescrum.org/`

6.5 e-Learning Resources

Links to online resources will be provided via Moodle[3] when relevant, but you are encouraged to find your own as well.

[3] `https://courses.ms.wits.ac.za/moodle/`

Apendix A: Notes

Notes on Tests/Exam

1. All tests/exams will be closed-book.
2. Your tests/exams may involve questions about your project, so keep that in mind when preparing for your assessments.
3. If you have queries regarding the marking of your test script, you must write a short paragraph to submit with your script for remarking, that describes why you believe that a specific question deserves more marks. If I've made a mistake adding this is not necessary, just bring the script to me.