School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
ELEN3009A — Software Development II

Course Brief and Outline — 2026

Academic Staff

Dr S.P. Levitt (course co-ordinator)
Room: CM3.237
Tel: 011 717 7209
Email: stephen.levitt@wits.ac.za

1 Course Background

This course forms part of the "Software Development" line of courses. It follows on from Software Development I and is a prerequisite for Software Development III. It deepens and broadens students' technical knowledge and skills with respect to software engineering and development.

2 Course Objectives

This course has three main objectives. It introduces the student to object-oriented modelling and design, programming using modern C++, and key software engineering technical practices such as unit testing and version control. More specialized software engineering courses flow from, and build on, the conceptual knowledge and practical skills developed in this course.

3 Course Outcomes

On successful completion of this course, the student is able to:

1. describe how C++ fits into the software development ecosystem and the trade-offs that it makes with respect to other programming languages;
2. perform object-oriented analyses of moderately complex software problems;
3. design and implement object-oriented solutions in C++ for such problems;
4. program in C++ in a modern, idiomatic fashion;
5. create a graphical user interface (GUI) using a game programming library;
6. utilise a test framework for software testing and verification;
7. use version control in a simple manner for tracking changes and sharing code;
8. use a formal notation to document a software design as well as an automated documentation tool.

4 Course Content

C++'s Place in the World
Compiled versus interpreted languages; strongly-typed versus weakly-typed languages; language popularity; programming paradigms

The Development Environment
The Integrated Development Environment (IDE); the build toolchain: pre-processor, compiler, linker; debugging

C++ Fundamentals
Primitive types, pointers and references; parameter passing; `const`; `static`, smart pointers, `auto`, range-based for loops, tuples

The C++ Standard Template Library
Containers; iterators; algorithms; vector and string

Programming Principles and Practices
Readable code, DRY principle, defensive coding, separation of concerns

Object-Oriented Analysis and Design
Levels of abstraction; interface and implementation; object-oriented analysis and design process; Tell, Don't Ask principle; Liskov substitution principle; code smells; modelling pitfalls

Object-Oriented Programming in C++
Classes and objects; constructors and destructors; copy construction and assignment; inheritance and polymorphism; aggregation.

Unit Testing
Automated unit testing, unit-test frameworks, test-driven development; writing good tests

Version Control
Git and GitHub, commits, fast-forward merges, pull requests, collaboration

Software Documentation
Unified Modelling Language (UML); class and sequence diagrams; technical documentation; extracting documentation from code

Error Handling
Exceptions; assertions; pre- and post-conditions; error handling schemes

Graphical User Interfaces
Game development library; graphics; event handling

5 Prior Knowledge Assumed

This course depends on content covered in Software Development I. In particular it assumes that students have a working knowledge of programming fundamentals (in C++). This includes variables; scope; flow control; raw pointers; functions, standalone classes, and the ability to write small programs to solve problems.

The prerequisites and co-requisites to register for this course are defined in the current *Rules & Syllabuses: Faculty of Engineering and the Built Environment*.

6 Assessment

6.1 Formative Assessment

Formative assessment takes place in the form of the course laboratories which need to be submitted for evaluation. Feedback is provided through comments on the source code submissions as well as discussions held during dedicated lab review sessions. The emphasis is on student participation and learning rather than whether solutions are correct or not. Formative assessment tasks do not count for marks.

Assessment Contributor	Duration (hours)	Component	Method & Weight	Calculator Type	Permitted Supporting Material
Laboratories	33	–	–	–	–

6.2 Summative Assessment

Assessment Contributor	Duration (hours)	Component	Method & Weight	Calculator Type	Permitted Supporting Material
Project	30	No	Rubric, 28%	–	–
Test	1	No	Marks, 26%	1	Reference sheets provided
Examination*	3	No	Marks, 44%	1	Reference sheets provided
Engagement	–	No	2%	–	–

*Note that the end-of-year examination requires a minimum of 35% in order to pass the course (termed a subminimum). This means that even if the average final mark for the course is 50% or above, a final examination mark of less than 35% does not qualify one to pass the course, as per the University Rules and Syllabuses.

Assessment Methods

Students will be assessed working *alone* (test and exam) and in *groups* (the project).

The project will require the student to work as part of a team to creatively identify, assess, formulate and solve a software development problem by using concepts, methods, tools and techniques introduced in this course. A marking rubric will accompany the project brief and this rubric will present the criteria for assessing the project outcomes.

The test is a written test which will take place during the semester. The material to be covered will be indicated by the lecturer.

The final exam will be time-tabled within the Oct/Nov examination period. All the material covered during the course (including lectures, laboratories, additional material, and the course project) is examinable.

The class test and final examination are *mandatory in-person* requirements for this course. Students will be required to write the test and attend all examinations (final exam/deferred/supplementary) on the Wits campus at the venue, and scheduled date and time, that is published by the lecturer, School or the Examination and Graduations Office.

A small percentage of marks count towards course engagement. The requirements for earning these marks will be communicated during the course.

7 Satisfactory Performance (SP) Requirements

For the purpose of Rule G.13 *satisfactory performance in the work of the class* means the completion of prescribed laboratory activities, submission of assignments, and writing of scheduled tests unless excused in terms of due procedure.

8 Teaching and Learning Process

8.1 Teaching and Learning Approach

This course is intentionally designed as *hybrid* course embracing the use of both online, and in-person, teaching and learning modes.

Table 1 shows the Graduate Attributes (GAs) associated with the course.

Table 1: GAs associated with the course and their corresponding level

Graduate Attribute (GA)	Level
GA 1: The laboratories, course project and exam require various modelling and algorithmic problems to be solved.	Developing
GA 3: The project requires the design and implementation of a small application. Engineering design is necessary for determining the overall architecture of the system as well as the design of individual components.	Developing
GA 5: Various software tools and frameworks need to be used appropriately during the course including a unit test framework, a graphics library, and version control.	Developing
GA 6a: The project needs to be documented thoroughly via code comments for a technical audience (software developers). This documentation is automatically extracted as an HTML report.	Developing
GA 8: The project involves students working together in pairs to deliver a software application. Labs involve both individual and teamwork aspects.	Developing
GA 11a: Work on the project, which lasts for a number of weeks, needs to be carefully planned in order to meet two interim deadlines and the final deadline.	Developing

8.2 Information to Support the Course

Supporting information for this course can be found on the course website. All the software development tools used during this course are freely available enabling students to use their own computing facilities.

There is no prescribed textbook for this course, although the textbook that is used in Software Development I covers some course material. Many good introductory texts on C++ programming are available in the Engineering and Geo/Maths libraries and at major bookshops. Additionally, there is an enormous amount of material that is accessible on the web. These resources can be used to supplement the course material.

Each of the following books are highly recommended if you wish to delve deeper into the language. *C++ Crash Course* is a concise and modern introduction to C++. *Effective Modern C++* is a very good "morality" guide — focusing on what you should do rather than presenting everything that you can do with C++. *C++ Coding Standards* and the *Core Guidelines* present important guidelines and considerations for C++ developers. Finally, *The C++ Programming Language* is by the creator of C++, Bjarne Stroustrup, and is the best reference for the language.

1. Josh Lospinoso. *C++ Crash Course: A Fast-Paced introduction*. No Starch Press, San Francisco, first edition, 2019
2. Scott Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, first edition, 2014
3. Herb Sutter and Andrei Alexandrescu. *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. C++ In-Depth Series. Addison-Wesley Professional, Berkeley, California, first edition, 2005; along with the C++ Core Guidelines
4. Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, fourth edition, 2013

8.3 Learning Activities and Arrangements

Lectures

The course content will be primarily delivered in the form of online videos and online supplementary material. Students are expected to watch the videos, and it will be taken that all material covered in this fashion has been viewed by students. In-person lectures will be highly interactive and will involve working through examples and problems.

Laboratories

There are a number of in-person laboratories associated with this course. *Doing the laboratories is indispensable for understanding the course material as they help to reinforce the concepts which are covered.* Teaching assistants and the course lecturer will be on hand to provide assistance during the laboratory sessions.

Project

The project brief will be handed out to the students during the course. It presents an opportunity to apply the course material. Students are required to work in pairs on the project over a number of weeks. Note that the School's policy on the timely submission of projects, as described in the *Red Book*, will be strictly enforced.

Consultation

Students are expected to consult with the lecturer or teaching assistants during the laboratory sessions or directly after lectures. Students should only contact the lecturer via email with regard to personal matters relating to the course.

9 AI Policy

The use of Generative AI and LLMs in this course is governed by the EIE Red Book and EIE GenAI Policy. This is a green course. AI may be used to help you understand code and software development/engineering concepts. Its use for reviewing and critiquing code is also encouraged.

AI should *not* be used for generating solutions, test code, or debugging. These are fundamental skills that you need to learn. LLMs are good tools when you have enough expertise to disagree with them, but in this course you are still developing your expertise.

Students are held solely accountable for the code they submit for the labs and the course project, meaning they must be able to explain and maintain it without the aid of AI. The use of AI tools is strictly prohibited during tests and exams.

10 Course Home Page

Further information and announcements regarding the course are posted on Ulwazi and/or the course home page: https://witseie.github.io/software-dev-2/

All students are expected to consult the course home page, and their email, at regular intervals.