Course Guide: Software Engineering (spring 2020)

General information

Course name:	Software Engineering
Code:	INF-32306
Credits:	6 ECTS (168 hours)
Language:	English
Schedule:	all afternoons during the first **seven** weeks of period 5 (March 16 – May 1) lectures on Monday 14:00–15:30 weeks 1–3 (C0067) and Thursday 14:00–15:30 weeks 1 (C0075) and 2 (C0062) rest of the time computer labs (PC0066) and group meetings
Coordinator/Examiner:	drs. M.R. Kramer
Other staff involved:	dr ir. A Kassahun, ir. M.A. Zijp, and possibly others

Keywords

software development, Object Orientation, testing, version management, UML, Java

Profile of the course

Many students will get involved in software related projects during their professional career. Often, they will have to communicate with software developers, either in the role of end users or as (formal) clients. In some cases, they will have to write their own programs, for instance to complete a thesis project that involves modeling, simulation, optimization, or large-scale data processing. For these reasons, it is essential for students to build up competencies in software development.
Software systems are developed to serve a specific purpose for specific groups of users. Such systems typically consist of many interacting components. Designing and implementing software systems goes beyond small-scale and ad-hoc programming. The techniques for building them are known under the term *Software Engineering*.
Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. In other words, it is the application of engineering to software. In this course we cover the process of developing a computer program from an initial idea to a tested and maintainable software system.

Learning outcomes

After successful completion of this course, students are expected to be able to:

- find occurrences of object-oriented (OO) software concepts in a given computer program (source code) or UML model
- write a software specification using UML
- assess adherence to specified software development practices
- design a software system using object-oriented (OO) techniques
- construct a working computer program in an OO programming language according to defined specifications
- formulate and execute functional tests for a software system
- operate a software version management system for sharing software components and documentation

Prerequisites

We expect that at the start of the course you to have elementary programming knowledge and skills (e.g. as taught in the course Programming in Python INF-22306)
Specifically you should be acquainted with the following concepts and techniques:

- variables, assignment, expressions, operators
- functions (and/or procedures, subroutines, methods) and parameters; also making your own functions
- control structures: at least: if, for, while
- objects and their properties (fields, variables) and operations (methods)
- arrays, including standard algorithms to traverse arrays (searching, summing, finding the largest element, etc.)

Study materials

Textbooks (both really used):

- Paul Deitel and Harvey Deitel: "Java, How to Program (Early Objects)" Global Edition 11/E (ISBN 9781292223858)
  [9th International Edition and 10th Early Objects Edition (both out of print) are still usable]
- Martina Seidl, Marion Scholz, Christian Huemer, Gerti Kappel: "UML@Classroom" (ISBN 978-3-319-12742-2; digitally available for free via WUR Library)

Software (available in PC-room):

- Eclipse: Integrated Development Environment (IDE) for Java
- TortoiseSVN: interface to Subversion (SVN; a version management system) or similar software for GIT (choice between SVN and GIT pending)
- any diagram drawing tool (used to be Microsoft Visio)

Materials on Brightspace:

- reading guides to the textbooks
- tutorials on Eclipse and specific operations within Eclipse (a.o. refactoring, version management)
- brief introduction "From Python to Java"
- self-assessment forms

Supplementary reading:

- John Hunt: "Agile Software Construction" (ISBN 978-1-85233-944-9; digitally available for free via WUR Library)

Activities

During the first two weeks, students acquire the relevant theoretical background and practical experience with the techniques and software development environments adopted for the course. Lectures introduce concepts of Object Oriented software development and the modeling (design) formalism UML (Unified Modeling Language). Dedicated tutorials and assignments introduce the programming language Java, software development techniques, and an integrated development environment (IDE) that supports these techniques. Students apply these techniques individually or in pairs during practical sessions.
During the rest of the course, groups of (ideally) eight students develop a computer program in a series of iterations. Each of these iterations starts with an investigation of functionalities to be incorporated in the software during the iteration, and ends with a working program that incorporates most – if not all – of the planned functionalities. Early in the fifth week, each group should produce a first release of their software. A final release is due at the end of the course.
In practice, customer requirements will change during the software development process. In this course, group supervisors play the role of customers and indicate new requirements after the first release. To accommodate the new requirements, most probably the structure of the software has to be adapted. Restructuring the code (refactoring) is an integral part of the software development process.
Groups use a version management system to track successive versions of their software. Because different team members may work simultaneously on the same parts of the software,

the version management system becomes also crucial to facilitate collaboration, especially to detect and solve conflicts between changes introduced by different team members.

Assessment

The grade for the course consists of two parts:
50% for the software product: final release of software built as group case

- interesting enough from a technical perspective
- functionalities implemented
- overall design
- class design
- coding

50% for the software process: steps in the process and techniques applied

- check in at version management system
- adding functionalities
- application of refactoring
- creating/updating UML models
- other documentation (e.g. user stories)

Note that there is no grade for fancy user interfaces.

Each individual student has to fill in two check-lists for all respective grading aspects, one check-list for the product and one check-list for the process. Each item in the check-lists has to be scored on a three-point scale (+, ±, and –) together with a specification of details.
The work, as well as the associated check-lists, will be assessed by course staff, in order to decide grades for both check-lists.
For each check-list, items scored too low will still be granted, up to an increase of 1.5 points and not exceeding a grade of 7.5. Items scored too high give an extra penalty equal to the difference in scores (after applying the previous clause).
In case of a severe imbalance between contributions to group work, the grade for the product can be individually adjusted.
To pass the course, both grades must be sufficient (i.e. at least 5.5).
When passing, the final grade of the course is the average of the two grades. When failing, the final grade is the lower of the two grades.
Partial scores are valid until the course starts in the same period in the next academic year.

Principal themes

Software Engineering life cycle

The field of *Software Engineering* has produced a number of ways to guide the software development process. In these so-called *life cycle models*, the same kinds of activities occur in different orders and with different interdependencies. Techniques for *software requirements analysis*, *architectural design*, *detailed design*, *coding*, and *testing* will be practiced in this course. For the relations between these techniques, we incorporate the main ideas of *Agile Software Development* in general and *Extreme Programming* in particular. One of the benefits of agile methods in software development is that they aim at frequent delivery of working software.

Object orientation

The key idea of all modern software development techniques is that software functions should be grouped around the kind of data that they process. Data is encapsulated in *objects* that have associated functionality. In real *Object-oriented* software development, this idea is extended with the notions of *inheritance* and *polymorphism*.

This course introduces the object-oriented design formalism UML (*Unified Modeling Language*) and the object-oriented programming language Java. Both UML and Java are widely used in software industry.
In building and organizing larger software systems, the notion of "*programming by contract*" takes encapsulation a step further. Java's mechanism of *interfaces* (not to be confused with user interfaces) plays an import role in implementing "programming by contract". In this course, we use interfaces as the primary mechanism for polymorphism.

Test driven development

Software should be tested before being delivered. Very often, tests are defined after the software has been written, or they not defined at all, especially when a project is running out of time. With *test driven development* the order is reversed: tests for a software feature should be defined before actually starting to incorporate that feature into the program. Software developers retain all tests, and execute the tests after every change they make to the software. Moreover, these tests give a formalized description of the intended (or understood) behavior of the software system or one of its components. Therefore, test driven development plays an important role in the activities of requirements analysis and design as well. From almost the start of the course to the end, we consistently apply *test driven development* in all development steps.

Version management

Software development is a very dynamic process. Even when a single person is working on a project, it is worthwhile to keep track of changes made during the development process. When developing software in a team, a system for tracking contributions and changes is almost indispensable. A *version management system* automates the bookkeeping of changes and software versions. All group work during the course is supported by a version management system.
Use of a version management system enables developers to return to an earlier version of the software, e.g. to find out which change introduced a newly discovered error. The change log that is maintained by the version management system helps tracing which changes were made for which reasons. Modern version management systems also offer support for merging independent changes made by different developers.

Refactoring

Many changes to software under development involve rather clerical (and often tedious) tasks, e.g. consistently renaming a variable or method (function), moving some code to another location, or adding a parameter to an existing method. Such tasks are supported by so-called *refactoring* mechanisms. Refactoring simplifies the process of adjusting the structure of software and thereby ensures a more flexible software development process.

Outline of the course

Week 1, week 2, and part of week 3: Introduction to UML, Java and development process; exercises with working environment.
Week 3/4: Start of group case; problem definition and first iteration.
Week 4/5: Second iteration leading to first release.
Rest of week 5 through week 7: Two more iterations.
During week 7 (or optionally early in week 8): Self-assessment.

Schedule

See separate spreadsheet.