COURSE DESCRIPTION CARD - SYLLABUS

Course name
Software Design and Modeling

Course

Field of study
Computing

Year/Semester
1/1

Area of study (specialization)
Software Engineering

Profile of study
general academic

Level of study
Second-cycle studies

Course offered in
English

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
30

Laboratory classes
30

Other (e.g. online)

Tutorials

Projects/seminars

Number of credit points

4

Lecturers

Responsible for the course/lecturer:
Bartosz Walter, Ph.D.

Responsible for the course/lecturer:

Prerequisites

Student should have basic knowledge on foundations of programming, including best practices and the design patterns. They should also be capable of continuous learning and knowledge acquisition from selected sources, as well as express the readiness for collaboarating in small teams.

Course objective

The objective for this course is to give the students knowledge on object-oriented software modeling and design, based on re-using commonly accepted best practices and design patterns elaborated and published in literature. Additionally, the course is expected to develop skills in evaluating the quality of software design and source code, and the use of selected mechanisms available in object-oriented programming languages.

Course-related learning outcomes

Knowledge

1. Students posesses well-grounded knowledge on the software system's life cycle.

2. Student posessess knowledge on selected methods, languages and notations used for developing software.

3. Student posesses knowledge on design patterns and best practices in software design

4. Student knows selected metrics and measurement methods for software quality characteristics (concerning the size, complexity, etc.)

Skills

1. Student can design a software system, using the mechanisms and features in object-oriented programming languages.

2. Student can evaluate the design quality of a software system.

3. Student can create useful model of a software system, using selected features of UML.

Social competences

1. Student can effectively collaborate in small teams.

2. Student enhances their knowledge, based on commonly available source, making a conscious selection of them.

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

The knowledge presented during the lecture will be verified two-fold: (i) by solving during the lectures in small teams two design case studies and discussing their pros and cons, and (ii) during the final examination (multilple-choice test that verifies the understading of the lectures). The two forms would be weighted 30:70, and the passing score is 50%. The list of examination problems will be provided during the last lecture within the course.

The skills acquired during laboratory classes will be verified by 3-4 group assignments, concerning the issues presented and discussed during the classes. The passing score is also 50%.

Programme content

1. Lecture: overview of methods and problems related to object-oriented prorgamming. Methods of software modeling. Unit testing. Measurements and metrics related to source code. Detailed overview of design patterns. Aspect-oriented programming. Functional programming. Inversion of control principle .

2. Laboratory classes: software modeling with CRC cards and elements of UML. Unit testing. Collecting software metrics and interpreting them. Selection and implementation of design patterns. The use of selected programming paradigms in practice. Implementation of the inversion of control in practice.

Teaching methods

1. Lecture: multimedia presentation, discussion

2. Laboratory classes: presentation supported by provided examples, programming the software and design assignments in groups, discussion

Bibliography

Basic

1. E. Gamma et al.: Design patterns. Elements of reusable OO software. Addison Wesley, 1995

2. R. C. Martin: Clean code. A Handbook of agile software craftmanship. Prentice Hall, 2008

3. B. Eckel: Thinking in Java (4th Edition). Prentice Hall, 2006

Additional

1. B. Meyer: Object-oriented software construction. Prentice Hall, 1994.

2. J. Backfield: Becoming functional. Steps for transforming into a functional programmer. O'Reilly Media, 2014.

Breakdown of average student's workload

	Hours	ECTS
Total workload	100	4,0
Classes requiring direct contact with the teacher	61	2,0
Student's own work (literature studies, preparation for laboratory classes/tutorials, preparation for tests/exam, project preparation) [1]	39	2,0

[1] delete or add other activities as appropriate