Foundations of software modelling - course description

General information

Course name	Foundations of software modelling
Course ID	11.3-WE-INFP-FounofSM-Er
Faculty	Faculty of Computer Science, Electrical Engineering and Automatics
Field of study	Computer Science
Education profile	academic
Level of studies	First-cycle Erasmus programme
Beginning semester	winter term 2019/2020

Course information

Semester	5
ECTS credits to win	6
Course type	optional
Teaching language	english
Author of syllabus	• dr inż. Łukasz Hładowski

Classes forms

The class form	Hours per semester (full-time)	Hours per week (full-time)	Hours per semester (part-time)	Hours per week (part-time)	Form of assignment
Lecture	30	2	-	-	Exam
Laboratory	30	2	-	-	Credit with grade
Project	15	1	-	-	Credit with grade

Aim of the course

- obtaining basic knowledge about software modelling

- familiarizing students with practical applications of software modelling for simple software systems

- familiarizing students with proper way of practical implementation of solution to simple problems using software design patterns

Prerequisites

Object-oriented programming, Software Engineering

Scope

Introductory issues. Background and history of modern modelling techniques. Unified process of application life cycle. System analysis and design. Object paradigm. Object modelling and its role in design of information systems. Class-Responsibility-Collaboration (CRC) diagrams. Software production processes.

*Introduction to Unified Modelling Language (UML) notation and diagrams.*Genesis and purpose of UML. Structural modelling. Basic notions and elements of object architecture: classes, objects, abstractions, encapsulation, inheritance, polymorphism, communication, relations and associations between objects. Static structural diagrams: class and object diagrams.Association modelling: aggregation, composition, generalization, specialization, dependencies and realization. Packages and subsystems. Types, interfaces and implementation classes. Implementation diagrams: component and deployment diagrams. Requirements and their specification. Use case diagrams.
Use case analysis: inclusion, extension, grouping and generalization. Behavioural modelling. Sequence and collaboration diagrams. Roles, messages and stimuli. Interactions and collaborations. Analysis of system states. State and activity diagrams. Flow transfer. Decisions. Concurrency. Signals and communication

Design patterns. Formulation of programming problems. Overview of most popular construction, structural and behavioural design patterns. Creational and testing patterns.

Practical issues. Work with use cases. General overview on design, deployment and testing. Presentation of dedicated UML design tools.

Teaching methods

lecture: brainstorm, discussion, practical tasks, conventional lecture

laboratory: brainstorm, working with source files, discussion, working in groups, practical tasks, conventional lecture

project: brainstorm, working with source files, discussion, working in groups, practical tasks, conventional lecture

Learning outcomes and methods of theirs verification

Outcome description	Outcome symbols	Methods of verification	The class form
Understands the need for unit tests and can implement them for simple cases. Can use simple programming tools for testing.		• a project	• Lecture • Project

Outcome description	Outcome symbols	Methods of verification	The class form
Can implement a fragment of simple system in chosen programming language using design patterns and proper object-oriented techniques.		• a project	• Project
Uses UML for description and formulation of solutions to programming problems.		• a project • a quiz • egzamin	• Lecture • Laboratory • Project
Can implement simple design patterns in chosen programming language. Knows disadvantages and advantages of a chosen pattern and can propose alternative solution.		• a project • a quiz • an examination test with score scale	• Lecture • Laboratory • Project
Can notice and describe advantages and disadvantages of proposed solution during the UML modelling phase.		• a project	• Project
Knows, understands and uses simple rules for software engineering.		• a project • a quiz • an examination test with score scale	• Lecture • Laboratory • Project
Can choose proper tools for software engineering.		• a quiz • an ongoing monitoring during classes	• Laboratory
Can notice and describe advantages and disadvantages of the proposed solution by source code analysis. Can refactor simple code.		• a project	• Lecture • Project

Assignment conditions

Lecture - a credit is given for obtaining a passing grade for all exams administered at least once per semester

Laboratory - to receive a final passing grade student has to receive passing grades for all tasks required by the curriculum.

Project - to receive a final passing grade student has to receive passing grades for all tasks and projects required by the curriculum..

Calculation of the final grade = lecture: 40% + laboratory: 20% + project: 40%

Recommended reading

1. Martin R.C.: Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008
2. Beck K.: Test Driven Development: By Example, Addison-Wesley Professional, 2002
3. Freeman E., Freeman E., Bates B., Sierra K.:Head First Design Patterns!, O'Reilly Media, 2004
4. UML @ Classroom, Seidl, M., Scholz, M, Springer International Publishing, 2015
5. Larman C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition), Prentice Hall, 2004

Further reading

1. Martin R.C., Martin M.: Agile Principles, Patterns, and Practices in C#, Prentice Hall, 2006
2. Way J.: Laravel Testing Decoded, Leanpub 2013

Notes

Modified by prof. dr hab. inż. Andrzej Obuchowicz (last modification: 27-10-2019 09:37)