



## COURSE DESCRIPTION CARD - SYLLABUS

Course name

Software engineering [S1S1E>IOP]

### Course

Field of study

Artificial Intelligence

Year/Semester

2/4

Area of study (specialization)

–

Profile of study

general academic

Level of study

first-cycle

Course offered in

English

Form of study

full-time

Requirements

compulsory

### Number of hours

Lecture

30

Laboratory classes

30

Other

0

Tutorials

0

Projects/seminars

0

### Number of credit points

4,00

### Coordinators

dr hab. inż. Mirosław Ochodek prof. PP  
miroslaw.ochodek@put.poznan.pl

### Lecturers

### Prerequisites

A student starting this course shall have basic knowledge concerning programming, tools, algorithms and data structures, object-oriented programming, computer systems architectures, and database systems. They should have the necessary skills to solve basic programming tasks and the ability to acquire knowledge from different sources.

### Course objective

1) Provide students with basic knowledge in the area of Software Engineering concerning IT project management, Requirements Engineering, systems modeling, software design, quality assurance (including software testing), and tools supporting software development (including version control systems). 2) Developing students' skills in solving simple problems regarding software designing, implementation, and testing; using tools supporting software development; modifying and using programming components. 3) Developing students' skills allowing them to work effectively as analysts/designers/software developers in software development teams following traditional or agile software development methodologies.

### Course-related learning outcomes

Knowledge:

1. Has basic knowledge regarding IT project management.
2. Has basic knowledge regarding Requirements Engineering (functional requirements, use cases, non-functional requirements).
3. Has basic knowledge regarding software modeling and design.
4. Has a basic knowledge regarding software verification and validation methods.

#### Skills:

1. Can effectively participate in Scrum events as a Developer (Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective).
2. Can specify functional and non-functional requirements.
3. Can create object models in UML notation (class model, state machine, sequence).
4. Can create test cases and automate them (unit tests, acceptance tests, and performance tests).

#### Social competences:

1. Is aware that the tools and programming libraries are subject to constant and frequent changes (e.g., by looking at the changes in the JUnit library or version control systems).
2. Knows the examples and understands the causes of information systems failures that led to serious financial and social losses or a serious health damage.
3. Can identify real-life problems that have commercial value and can be solved through the development of information systems.

### Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

#### Formative assessment:

- a) in terms of lectures: based on answers to questions and participation in quizzes during classes lectures
- b) in the scope of the laboratory: based on the assessment of the current progress of tasks in progress lab

#### Summary rating:

In terms of assessing educational outcomes regarding acquired skills and social competences (mainly grade from laboratory classes):

- a) depending on the level of completion of tasks in individual laboratory classes, the student may receive 0 or 10 points. A student absent from classes may make up classes at another time or complete the tasks at home with the teacher's consent. Each student can score from 0 to 120 in total points.
- b) during the semester, students carry out a group project (3-5 people) according to the recommendations of the Scrum methodology.

The project consists of two sprints (iterations). In each sprint, the team can achieve from 0 to  $n \cdot 100$  (where  $n$  is the number of people in the team) points depending on the degree of task implementation. Each member

team can receive a maximum of 100 points per sprint, giving a total maximum of 200 points.

Based on the sum of points obtained, the final grade is determined as follows

frets:

- $\geq 280$  - 5.0
- $< 250, 280$  - 4.5
- $< 220, 250$  - 4.0
- $< 190, 220$  - 3.5
- $< 160, 190$  - 3.0
- less than 160 - 2.0

In terms of learning outcomes regarding acquired knowledge:

- a) during lectures, students solve quizzes and short problem-solving tasks or they take part in the quiz. For providing an acceptable solution (depending on its form and character) the student receives 1%.
- b) multiple choice test including 25 multiple-choice questions (one correct answer) or questions with possibly one or more correct answers (the type of question is explicitly indicated in the test). Behind Answering the question correctly, the student receives 1 point. Points are converted to a scale percentage.

Based on the percentage points obtained (from the choice test and during the lecture) it is determined is the final grade according to the scale:

- >= 90% - 5.0
- <80%, 90%) - 4.5
- <70%, 80%) - 4.0
- <60%, 70%) - 3.5
- <50%, 60%) - 3.0
- less than 50% - 2.0

## Programme content

The course program covers the following topics:

- Role of software development in the modern world, a vision of an IT project, consequences of software failure, and the scope of Software Engineering
- Software configuration management (including version control systems - Git and Subversion, automatic software building tools - Apache Ant and Apache Maven, continuous integration practices, and the basics of configuration management for run-time environments (containers)
- Functional requirements (including use cases)
- Non-functional requirements (including the ISO/IEC 25010 standard)
- Software modeling and analysis (including the UML notation)
- Software design (including Design Patterns)
- Software architecture
- Project management methodologies (Scrum and PRINCE2)
- Software quality management (including measurement in the software development process)
- Software testing (unit, integration, acceptance, and non-functional testing)

The program of laboratory classes covers the following topics:

- Risk assessment in IT projects
- Software configuration management tools, e.g., Git, Apache Ant, Apache Maven
- Documenting functional requirements with use cases
- Documenting non-functional requirements
- Modeling software systems with the UML notation
- Designing software using Design Patterns
- Software testing, including unit and performance testing
- Running a mini-project according to the recommendations of the Scrum framework.

## Course topics

Lecture program includes the following topics:

Introduction to software engineering and continuous integration (CI)  
 Software configuration management (Git)  
 Functional requirements in IT projects  
 Non-functional requirements  
 Software modeling (UML notation)  
 Project management methodologies  
 Design patterns  
 Unit and integration testing  
 Software architecture  
 Acceptance testing  
 Non-functional testing  
 Code quality measurement  
 IT project planning  
 Introduction to DevOps

Laboratory program includes the following topics:

Introductory classes and risk management in IT projects  
 Integrated development environments (IDE) and technical code documentation  
 Software building and continuous integration - software building tools (Apache Ant, Apache Maven) and continuous integration service (GitHub Actions)  
 Git version control system and GitHub repository hosting service

Functional requirements - defining requirements in the form of use cases  
 System modeling using UML notation - class and sequence diagrams  
 Design patterns - identification of patterns in Java standard library and code refactoring  
 Mini-project implementation using Scrum methodology - two sprints utilizing learned methods and tools  
 Unit testing using JUnit and Mockito  
 Acceptance testing - manual tests and automation using Selenium  
 Performance testing using JMeter

## Teaching methods

The mini-project is organized according to the teaching method described in the following paper:  
 Ochodek, Mirosław. "A Scrum-Centric Framework for Organizing Software Engineering Academic Courses." In Towards a Synergistic Combination of Research and Practice in Software Engineering, pp. 207-220. Springer, Cham, 2018.

Other teaching methods used:

- a) lecture: multimedia presentation, presentation illustrated with examples given on the whiteboard, solving problems, case studies.
- b) laboratory: problem-solving, practical exercises, discussion, teamwork, multimedia show, workshops, demonstration.

## Bibliography

Basic

1. I. Sommerville, Software Engineering, 9th ed., Pearson Educatin, 2011.
2. K. Schwaber, J. Sutherland, The Scrum Guide, <http://www.scrumguides.org>, (online), 2020.

Additional

1. Ochodek, Mirosław, J. Nawrocki, and K. Kwarciak. Simplifying effort estimation based on Use Case Points. Information and Software Technology 53.3 (2011): 200-213.
2. Kopczyńska, Sylwia, Jerzy Nawrocki, and Mirosław Ochodek. An Empirical Study on Catalog of Non-functional Requirement Templates: Usefulness and Maintenance Issues. Information and Software Technology (2018).
3. Nawrocki, Jerzy, et al. Agile requirements engineering: A research perspective. International Conference on Current Trends in Theory and Practice of Informatics. Springer, Cham, 2014.

## Breakdown of average student's workload

	Hours	ECTS
Total workload	100	4,00
Classes requiring direct contact with the teacher	60	2,50
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	40	1,50