KARTA OPISU MODUŁU KSZTAŁCENIA

Nazwa modułu/przedmiotu **Projektowanie i modelowanie oprogramowania**		Kod **1010512321010517859**
Kierunek studiów **Informatyka**	Profil kształcenia (ogólnoakademicki, praktyczny) **(brak)**	Rok / Semestr **1 / 2**
Ścieżka obieralności/specjalność **Software Engineering (Inżynieria**	Przedmiot oferowany w języku: **angielski**	Kurs (obligatoryjny/obieralny) **obligatoryjny**
Stopień studiów: **II stopień**	**Forma studiów** (stacjonarna/niestacjonarna) **stacjonarna**	
Godziny Wykłady: **30** Ćwiczenia: **-** Laboratoria: **30** Projekty/seminaria: **-**		Liczba punktów **5**
Status przedmiotu w programie studiów (podstawowy, kierunkowy, inny) **(brak)**	(ogólnouczelniany, z innego kierunku) **(brak)**	

Obszar(y) kształcenia i dziedzina(y) nauki i sztuki	Podział ECTS (liczba i %)
nauki techniczne	**4 100%**
nauki techniczne	**4 100%**

Odpowiedzialny za przedmiot / wykładowca:

Bartosz Walter
email: bartosz.walter@cs.put.poznan.pl
tel. 616652980
Faculty of Computing
ul. Piotrowo 3 60-965 Poznań

Wymagania wstępne w zakresie wiedzy, umiejętności, kompetencji społecznych:

1	**Wiedza:**	Student starting this module should have a basic knowledge of software engineering and object-oriented design.
2	**Umiejętności:**	Should have skills to design and implement of simple software systems and skills of solving basic problems related to requirements analysis, creating software specification, designing systems and skills that are necessary to acquire information from given sources of information.
3	**Kompetencje społeczne**	Student should understand the need to extend his/her competences / has the willingness to work in a team. In addition, in respect to the social skills the student should show attitudes as honesty, responsibility, perseverance, curiosity, creativity, manners, and respect for other people.

Cel przedmiotu:

1. Provide students with knowledge on OOP, in particular the role, responsibility and relationships of objects
2. Present methods of evaluating design quality of object-oriented systems with use of OO metrics and code smells
3. Develop students? teamwork skills in the context of designing software systems
4. Present unit testing as a method for verification if objects properly fulfill their responsibilities
5. Present design patterns as a reusable schemas leading to improving quality of object-oriented design. Teaching students using design patterns in implementing software systems.
6. Present software refactoring as a technique of improving internal quality of software systems.

Efekty kształcenia i odniesienie do kierunkowych efektów kształcenia

Wiedza:

1. student has well-established theoretical knowledge of computer systems architecture, software design, software testing and verification, software engineering - [K_W4+++]
2. student has detailed theoretical knowledge related to selected areas of computer science creating software architecture, documenting system architecture, evaluation of architecture, modeling software, designing software, testing and verifying software - [K_W5+++]
3. student has knowledge regarding trends and the most important new developments in computer science and related disciplines - [K_W6+]
4. student has basic knowledge regarding life-cycle of software or hardware systems - [K_W7+]
5. knows the fundamental methods, techniques and tools employed to solve complex engineering tasks in a selected area of software architecture, software modeling and design, software testing and verification - [K_W8+++]

Umiejętności:

1. student is able to acquire, combine, interpret and evaluate information from literature, databases and other information sources (in mother tongue and English); draw conclusions, and formulate opinions based on it. - [K_U1]

2. student is able to plan and arrange self-education process - [K_U5+]

3. student has language skills at B2+ level in accordance with the requirements set out for level B2+ Common European Framework of Reference for Languages - [K_U6+]

4. student is able to employ analytical, simulation, and experiment methods to formulate and solve engineering tasks and basic research problems - [K_U9+]

5. student is able to combine knowledge from different areas of computer science (and if necessary from other scientific disciplines) to formulate and solve engineering tasks; and use system approach that also incorporates nontechnical aspects - [K_U10+]

6. student is able to formulate and test hypotheses regarding engineering problems and basic research problems - [K_U12+]

7. student is able to assess usefulness and possibility of employing new developments (methods and tools) and new IT products - [K_U13++]

8. student is able to develop an object-oriented model of a simple software system (e.g., in UML notation) - [K_U17+++]

9. student is able to assess software architecture from the perspective of non-functional requirements - [K_U18+]

10. student is able to effectively participate in software inspections - [K_U19+]

11. student is able to propose enhancements (improvements) to existing technical solutions - [K_U21+]

12. student is able to evaluate usefulness of methods and tools (also to identify their limitations) used to solve engineering tasks, i.e., building IT systems or their components - [K_U24+]

13. student is able to design (according to a provided specification which includes also non-technical aspects) a complex device, an IT system, or a process; and is able implement it (at least partially) using appropriate methods, techniques, and tools (including adjustment of available tools or developing new ones) - [K_U27+++]

Kompetencje społeczne:

1. student understands that knowledge and skills related to computer science quickly become obsolete - [K_K1+]

2. student is able to correctly assign priorities to own tasks and tasks performed by others - [K_K6+]

Sposoby sprawdzenia efektów kształcenia

Formative assessment:

a) lectures:

? based on the answers to the questions which test understanding of material presented on the lectures

b) laboratory classes / tutorials / projects / seminars:

? based on the assessment of the tasks done during classes and as a homework

Summative assessment:

a) verification of assumed learning objectives related to lectures:

? assessment of knowledge and skills, examined by a written test with multiple choices and problem questions. Student can gain 10.0 pts; passing limit is 5.0 pts

? discussing the results of the examination

b) verification of assumed learning objectives related to laboratory classes / tutorials / projects / seminars:

? assessment of student?s preparation to particular laboratory classes and assessment of student?s skills needed to realize tasks on these classes

? continuous assessment of student?s work during classes ? rewarding ability to use learned principles and methods

? assessment of projects realization, including ability to work in team

Possibility to gain additional points by activity on classes:

? elaboration of additional aspects regarding the subject

? effectiveness of applying acquired knowledge to solve problems

? ability to cooperate with the team during solving problems

? providing additional remarks for the lecturer how to improve teaching materials

? elaboration of an outstanding solution to an assignment ? for use as a case-study

? highlighting the problems with students? perception to improve the teaching process

Treści programowe

The program of the lecture:

The concept of objects and object-oriented perception. Mechanisms of object-oriented programming. Object-oriented languages vs. object-oriented design. Roles of different types of objects in design. Criteria for evaluation of object-oriented design. Metrics and their interpretation. Unit testing. Mock objects. Design patterns ? idea, description, categories. Overview of the catalogue of design patterns, with description of goal, description, participants and consequences ? for each of them. The code decay phenomenon ? reasons, symptoms, consequences. High-level evaluation of design quality with code smells. Methods of identification of code smells. Overview of selected refactorings. Verification methods of refactorings. Aspect-oriented programming and its implementation in different technologies. AspectJ as an aspect-oriented language. Inversion of Control and Dependency Injection.

The course consists of fifteen 2-hour laboratory classes and it starts with an instructional session at the beginning of a semester. Students work individually or in teams of 2-4.

The program of laboratory classes:

Creating preliminary design with CRC cards. Analysis and evaluation of the CRC design. Assigning responsibility to objects. Measuring software with OO metrics. Analysis and interpretation of OO metrics. Implementing unit tests. Applying mock objects in unit tests. Selection and application of appropriate design patterns in design problems. Identification of code smells in code. Comparison of metrics and code smells as tools for evaluation of design quality. Applying software refactorings (both manually and with tools support). Implementation of a simple aspect-oriented program and use of selected capabilities of AspectJ. Design and implementation of a simple program based on a Inversion of Control concept.

Literatura podstawowa:

Literatura uzupełniająca:

Bilans nakładu pracy przeciętnego studenta

Czynność	Czas (godz.)
1. participating in laboratory classes / tutorials: 15 x 2 hours, 2. 2. consulting issues related to the subject of the course; especially related to t laboratory classes and projects, 3. implementing, running and verifying software application(s) (in addition to laboratory classes) 4. participating in lectures 5. studying literature / learning aids (10 pages = 1 hour), 60 pages 6. discussing the results of the examination 7. preparing to and participating in exams: 5 hours + 2 hours	30 10 16 30 6 1 7

Obciążenie pracą studenta

forma aktywności	godzin	ECTS
Łączny nakład pracy	100	4
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	73	3
Zajęcia o charakterze praktycznym	56	2