STUDY MODULE DESCRIPTION FORM

Name of the module/subject	Code
Software Design and Modelling	**1010512321010517859**

Field of study	Profile of study (general academic, practical)	Year /Semester
Computing	**general academic**	**1 / 2**
Elective path/specialty: **Software Engineering**	Subject offered in: **Polish**	Course (compulsory, elective): **obligatory**
Cycle of study: **Second-cycle studies**	Form of study (full-time,part-time): **full-time**	
No. of hours: Lecture: **30** Classes: **-** Laboratory: **30** Project/seminars: **-**		No. of credits: **5**
Status of the course in the study program (Basic, major, other): **major**	(university-wide, from another field): **from field**	
Education areas and fields of science and art: **technical sciences** **Technical sciences**		ECTS distribution (number and %): **5 100%** **5 100%**

Responsible for subject / lecturer:

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

Prerequisites in terms of knowledge, skills and social competencies:

1	**Knowledge**	Student starting this module should have a basic knowledge of software engineering and object-oriented design.
2	**Skills**	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	**Social competencies**	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.

Assumptions and objectives of the course:

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.

Study outcomes and reference to the educational results for a field of study

Knowledge:

1. student has well-established theoretical knowledge concerning key elements of computer science - [K2st_W2]
2. student has extensive, detailed knowledge related to selected areas of computer science - [K2st_W3]
3. student has extensice, detailed knowledge related to processes in software- and hardware-based information systems - [K2st_W5]
4. student knows advanced methods, techniques and tools used for solving complex engineerial tasks and conducting research in selected areas of computer science - [K2st_W6]

Skills:

1. student, on defining and solving engineerial tasks, is able to combine knowledge from various areas of computer science (and other areas, if needed) and apply systematic approach that includes also non-technical aspects - [K2st_U5]

2. student is able to evaluate the usefulness of new advances (methods and tools) and new IT products - [K2st_U6]

3. student is able, using also new methods, solve complex information tasks, including non-standard or research-related ones - [K2st_U10]

4. student, based on a specification that involves also non-technical aspects, is able to design a complex device, system or process, and implement the project (at least partially), by using appropriate methods, techniques and tools, by adjusting them or by creating new ones - [K2st_U11]

Social competencies:

1. student understands that knowledge and skills related to computer science quickly become obsolete - [K2st_K1]

2. student understands the relevance of applying new findings and knowledge of computer science in solving research- and practical problems - [K2st_K2]

Assessment methods of study outcomes

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

Course description

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.

Basic bibliography:

1. E. Gamma et al.: Design patterns. Elements of reusable object-oriented software. Addison-Wesley, 1994.

2. M. Fowler: Refactoring. Improving design of existing software. Addison-Wesley, 1999.

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

Additional bibliography:

1. B. Meyer: Object-oriented software construction (2nd Edition). Prentice Hall, 2000.

Result of average student's workload

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

Student's workload

Source of workload	hours	ECTS
Total workload	100	5
Contact hours	62	2
Practical activities	56	2