SYLLABUS

1. Information regarding the programme

1.1 Higher education institution	**Babeş Bolyai University**
1.2 Faculty	**Faculty of Mathematics and Computer Science**
1.3 Department	**Department of Computer Science**
1.4 Field of study	**Computer Science**
1.5 Study cycle	**Master**
1.6 Study programme / Qualification	**Component-Based Programming**

2. Information regarding the discipline

2.1 Name of the discipline	**Software design**						
2.2 Course coordinator	**Prof.PhD. Bazil Parv**						
2.3 Seminar coordinator	**Prof.PhD. Bazil Parv**						
2.4. Year of study	**1**	2.5 Semester	**2**	2.6. Type of evaluation	**E**	2.7 Type of discipline	**compulsory**
2.8 Code of the discipline	**MME8065**						

3. Total estimated time (hours/semester of didactic activities)

3.1 Hours per week	**4**	Of which: 3.2 course	**2**	3.3 seminar/lab	**1s+1pr**
3.4 Total hours in the curriculum	**56**	Of which: 3.5 course	**28**	3.6 seminar/lab	**28**
Time allotment:					Hours
Learning using manual, course support, bibliography, course notes					**28**
Additional documentation (in libraries, on electronic platforms, field documentation)					**28**
Preparation for seminars/labs, homework, papers, portfolios and essays					**60**
Tutorship					**14**
Evaluations					**14**
Other activities:					-

3.7 Total individual study hours	**144**
3.8 Total hours per semester	**200**
3.9 Number of ECTS credits	**8**

4. Prerequisites (if necessary)

4.1. curriculum	• Fundamentals of programming • Object-oriented programming • Programming paradigms
4.2. competencies	• Average programming skills

5. Conditions (if necessary)

5.1. for the course	• Videoprojector, Internet access
5.2. for the seminar /lab activities	• Computers, Internet access, UML tool

6. Specific competencies acquired

Professional competencies	• Understanding of the software design from the engineering perspective; • Understanding of the software design concepts and principles • Understanding of the software design process and its activities; • Proficient use of tools and languages specific to software systems development • Knowing the specifics of main architectural and design patterns and how to apply them to specific projects.
Transversal competencies	• Professional communication skills; concise and precise description, both oral and written, of professional results, • Independent and team work capabilities; able to fulfill different roles • Antepreneurial skills;

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• Know and understand fundamental concepts of software design. • Be able to apply the appropriate architectural and design patterns to different programming projects
7.2 Specific objective of the discipline	At the end of the course, students • know the main concepts and principles of software design • have a good understanding of the following terms: software architecture definition(s), architectural styles and models, architecture definition language(s); detailed design; design pattern, construction design; • learn the importance of architectural and detailed design and how to use tools for these tasks; • know several software system types (taken from real-world applications) and the best recommended architectural styles and design patterns.

8. Content

8.1 Course	Teaching methods	Remarks
1. *Introduction to software engineering design*. Motivation and general design concepts. Overview of the software engineering design. Functional and non-functional requirements. Quality attributes. Constraints	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
2. *Software design process*. Main phases: architectural design, detailed design, construction design, data design, UI design. Inputs and deliverables	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
3. *Software architecture 1*. Definitions. Principles. Fundamentals of requirements engineering. Designing the software architecture	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
4. *Software architecture 2*. Architectural styles and patterns - overview and history. Architectural patterns for data-centered systems	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
5. *Software architecture 3*. Architectural patterns for data-flow systems	• Interactive exposure • Explanation • Conversation • Didactical demonstration	

6. *Software architecture 4*. Architectural patterns for distributed systems	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
7. *Software architecture 5*. Architectural patterns for interactive and hierarchical systems	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
8. *Detailed design 1*. Overview of the detailed design. Structural and behavioral design of components. Design principles	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
9. *Detailed design 2.* Creational design patterns: Abstract Factory, Factory Method, Builder, Prototype, Singleton.	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
10. *Detailed design 3.* Structural design patterns: Adapter, Bridge, Composite, Façade	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
11. *Detailed design 4.* Behavioral design patterns: Iterator, Observer, Strategy, Template Method	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
12. *Construction design*. Flow-, state-, and table-based construction design. Programming design language, styles, and quality evolution.	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
13. *Design evolution 1*. Architecture refactoring. Detailed design refactoring	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
14. *Design evolution 2*. Construction design refactoring	• Interactive exposure • Conversation	

Bibliography

1. BASS, L., CLEMENTS, P., KAZMAN R.: *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003
2. FOWLER, MARTIN: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999
3. KRUCHTEN, PH.: *Architectural Blueprints – The 4+1 View Model of Software Architecture*, IEEE Software 12 (6), 1995, pp. 42-50.
4. MARTIN, ROBERT CECIL: *Agile software development: principles, patterns, and practices*, Pearson Education, 2002
5. McCONNELL, STEVE: *Code Complete*, 2nd ed., Microsoft Press, 2004
6. OTERO, C.E.: *Software Engineering Design*, CRC Press, 2012.
   site: http://softwareengineeringdesign.com/Default.htm
7. SHAW, M.: *The Coming-of-Age of Software Architecture Research*, in Proc. of the 23rd ICSE, IEEE Comp. Soc. 2001, 656, [http://www.cs.cmu.edu/afs/cs.cmu.edu/project/vit/ ftp/pdf/shaw-keynote-rev.pdf]
8. SHAW, M., GARLAN, D.: *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

8.2 Seminar / laboratory	Teaching methods	Remarks
1. Administrivia	Conversation, debate, case studies, presentations	Seminar is organized as a total of 14 hours – 2 hours every other week
2. Establishing the target application. First miniproject started	Conversation, debate, case studies, examples	
3. Work on miniproject 1	Exposure, debate, case studies, examples	

4. Miniproject 1 due. Second miniproject started	Exposure, debate, case studies, examples	
5. Work on miniproject 2	Exposure, debate, case studies, examples	
6. Miniproject 2 due. Detailed design issues	Exposure, debate, case studies, examples	
7. Final review and project evaluation	Exposure, live demos	

Bibliography

Students will serch and use software design documentation

- on the department server (win/labor/Romana/master/SED)
- on the web, using main CS databases

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- This course follows the IEEE and ACM Curriculla Recommendations for Software Engineering studies;
- Courses with similar content are taught in the major universities in Romania offering similar study programs;
- Course content is considered very important by the software companies for improving average software development skills

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Course	• knowing the basic concepts of software design • applying different architectural styles and design patterns to different problem domains	Written exam	40%
10.5 Seminar/lab activities	• be able to study and review literature regarding software design • be able to solve a problem using different architectural and design patterns • be able to evaluate a software design	• Miniproject 1 work • Miniproject 2 work • Seminar/lab attendance • Default	20% 20% 10% 10%
10.6 Minimum performance standards			
• At least grade 5 (from a scale of 1 to 10) at written exam and each miniproject work.			

Date	Signature of course coordinator	Signature of seminar coordinator
April 19, 2018	Prof.PhD. Bazil PARV	Prof.PhD. Bazil PARV
Date of approval		Signature of the head of department
.....		Prof.PhD. Anca ANDREICA