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	**Bachelor**
1.6 Study programme / Qualification	**Computer Science**

2. Information regarding the discipline

2.1 Name of the discipline (en) (ro)			**Design Patterns**				
2.2 Course coordinator			**Assoc. Prof. PhD. Arthur Molnar**				
2.3 Seminar coordinator			**Assoc. Prof. PhD. Arthur Molnar**				
2.4. Year of study	**3**	2.5 Semester	**6**	2.6. Type of evaluation	**C**	2.7 Type of discipline	**Opt**
2.8 Code of the discipline		**MLE8115**					

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

3.1 Hours per week	5	Of which: 3.2 course	2	3.3 seminar/laboratory	1 L + 2 PR
3.4 Total hours in the curriculum	60	Of which: 3.5 course	24	3.6 seminar/laboratory	36

Time allotment:	hours
Learning using manual, course support, bibliography, course notes	15
Additional documentation (in libraries, on electronic platforms, field documentation)	15
Preparation for seminars/labs, homework, papers, portfolios and essays	15
Tutorship	15
Evaluations	5
Other activities:	-

3.7 Total individual study hours	65
3.8 Total hours per semester	125
3.9 Number of ECTS credits	5

4. Prerequisites (if necessary)

4.1. curriculum	• Fundamentals of Programming • Object Oriented Programming
4.2. competencies	• Good programming skills in Java or C#

5. Conditions (if necessary)

5.1. for the course	• Lecture hall with projector
5.2. for the seminar /lab activities	• Computers with installed IDE for Java/C# development

6. Specific competencies acquired

Professional competencies	C1.1 Description of programming paradigms and of language specific mechanisms, as well as identification of syntactic and semantic differences. C1.2 Explanation of existing software applications, on different levels of abstraction (architecture, classes, methods) using adequate basic knowledge. C1.3 Elaboration of adequate source code and testing of components in a given programming language, based on given specifications. C2.1 Identify adequate methodologies to develop software systems C2.3 Use methodologies, specification and IDEs to develop software systems C2.5 Implement dedicated software systems C4.3 Identify models and methods to solve real-life problems
Transversal competencies	CT1 Application of efficient and rigorous working rules, manifest responsible attitudes towards the scientific and didactic fields, respecting professional and ethical principles. CT3 Use of efficient methods and techniques for learning, information, research and development of abilities for knowledge exploitation, for adapting to the needs of a dynamic society and for communication in a widely used foreign language.

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

7.1 General objective of the discipline	• Enhance students' understanding of software design concepts through a pragmatic approach. • Provide students with an environment in which they can explore the usage and usefulness of software design concepts in various business scenarios. • Induce a realistic and industry driven view of software design concepts such as design patterns and their inherent benefits
7.2 Specific objective of the discipline	• Give students the ability to explore various object-oriented programming languages. • Improve the students' abilities to tackle business requirements. • Enhance the students' understanding of business needs and business value. • Provide students with insights into the way of working towards achieving high quality software.

8. Content

8.1 Course	Teaching methods	Remarks
1. OOP Principles Recap: Cover main OOP principles such as encapsulation,	description, explanation,	-

polymorphism, cohesion, coupling, aggregation, composition using well known languages (Python, C++, Java, C#, etc.)	example, case studies, dialogue, debate	
2. SOLID principles: base principles of high-quality software: Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion		
3. Creational Patterns (Factory, Builder, Prototype, Singleton)		
4. Structural Patterns (Adapter, Bridge, Composite)		
5. Structural Patterns (Decorator, Facade, Flyweight, Proxy)		
6. Behavioural Patterns (Chain of Responsibility, Command, Iterator)		
7. Behavioural Patterns (Mediator, Memento, Observer)		
8. Behavioural Patterns (State, Strategy, Template, Visitor)		
9. Antipatterns, Dark Patterns in the UX		
10. Architectural Patterns (MVVM, MVP, MVC)		
11. Enterprise Integration Patterns (selection)		
12. Examination		
Bibliography 1. M. Fowler – Patterns of Enterprise Application Architecture, Aison Wesley, 2003 2.E. Freeman, E. Freeman, B. Bates – Head First Design Patterns, Oreilly, 2004 3. E. Gamma, R. Helm, R.Johnson, J. Vlissides – Design Patterns Elements of Reusable Object-Oriented Software, Addison Wesley, 1995 4. Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns, Addison Wesley, 2003.		
8.2 Seminar / laboratory	Teaching methods	Remarks
1. Introduction. OOP recap. Advanced UML elements.	Explanation, dialogue, case studies	-
2. SOLID principles		
3. Creational Design Patterns		
4. Structural Design Patterns		
5. Behavioural Design Patterns		
6. Antipatterns, Architectural Patterns		
7. Final project turn-in		
Bibliography 1. M. Fowler – Patterns of Enterprise Application Architecture, Aison Wesley, 2003 2.E. Freeman, E. Freeman, B. Bates – Head First Design Patterns, Oreilly, 2004 3. E. Gamma, R. Helm, R.Johnson, J. Vlissides – Design Patterns Elements of Reusable Object-Oriented Software, Addison Wesley, 1995 4. Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns, Addison Wesley, 2003.		

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

- The course respects the IEEE and ACM Curriculla Recommendations for Computer Science studies.
- The course exists in the studying program of all major universities in Romania and abroad.
- The content of the course is considered important within the software industry for acquiring advanced programming skills.

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Lecture	Team presentations during the semester.	Technical quality, thoroughness of presentation.	25%
10.4 Lecture	Oral examination in the form of design pattern exemplification in open-source software.	Level of technical complexity and suitability of the presented pattern examples.	50%
10.5 Seminar/lab activities	Final project: design pattern application.	Number and variety of implemented patterns, technical quality, and presentation.	25%
10.6 Minimum performance standards			

- Students must observe the standards of academic integrity.
- Students must show good understanding of traditional and architectural design patterns, be able to identify them in complex, real-world applications and identify when their application can result in tangible improvements to software quality.
- A minimum passing grade is defined by attaining at least 50% (5/10) points from the total represented in the table above.

Date

Signature of course coordinator

Assoc. Prof. PhD. Arthur Molnar

Signature of seminar coordinator

Assoc. Prof. PhD. Arthur Molnar

Signature of the head of department

Assoc. Prof. PhD. Adrian Sterca