Politehnica University
FILS
Prof. Luca Dan Serbanati
Software Design Techniques, IV, 1, 2C, 0S, 1L, 1P, E, 4

Course description	This course is concerned with the design of complex software systems, that is the architecture of the overall software product and then the design of objects that populate the run-time environment. In software systems the building blocks must be carefully verified and integrated to ensure that the resulting applications are robust and maintainable. The necessity to integrate, reuse, and maintain large collections of software components has led to important challenges for software engineers which, in turn, resulted in the elaboration of various component models, design patterns, and integration mechanisms. Design patterns are an other issue of the course. A Design Pattern is a best practice solution to a reoccurring problem in some context. Design patterns encapsulate proven, reusable solutions to common design issues. The study of design patterns can help advance the technical expertise of software professionals. The course includes design pattern presentation at both architectural and low levels. Architecture of software is a collection of design decisions that are expensive to change. It delineates the structure of the system, showing its major aspects that need to be understood in order to understand how the system is put together. In this course the students will acquire the concepts and techniques that will enable them to understand, analyze, maintain, and improve the architecture of large software systems. Several example problems will be studied to investigate the development of good design decisions at either architectural or object level. Students are expected to practice their use in modelling, designing, building, and validating practical, high-quality software systems.
Prerequisite(s)& Corequisite(s)	• "Software Development Methods" course (including the development of a non trivial software project) • In-depth knowledge of UML • Worthy experience of programming in Java or C++
Textbook(s) and web materials	1. M. Shaw, D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996, ISBN: 0131829572 2. M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002 3. P. Kuchana, *Software architecture design patterns in Java*, Auerbach Publications, 2004, ISBN 0-8493-2142-5 4. G. Hohpe, B. Woolf, *Enterprise Integration Patterns*, Addison-Wesley, 2004 5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* , Addison-Wesley, 1995 6. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-oriented software architecture. A system of patterns*, John Wiley&Sons, 1996. 7. http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
Course objectives	The overall goal of this course is to gain knowledge of software design best practices. By the end of the semester, students will be able to: 1. describe what is meant software design and describe its reason for existence in the software life-cycle, 2. understand add apply component coupling and cohesion in an OO design 3. describe the main design patterns, 4. exercise the design patterns, 5. identify which design decisions are expensive to change, 6. describe the main architectural styles,

	7. analyze and evaluate architectural qualities, 8. demonstrate a working knowledge of the worth of software reusability, 9. have the academic background to follow current literature in software patterns.
Topics covered	**I. Introduction** • Reviewing the main aspects of OOAD and UML (2h) **II. Design Patterns** • Fundamental Patterns: Delegation, Interface, Marker Interface, Proxy (2h) • Creational Patterns: Factory Method, Builder, Prototype, Singleton (3h) • Partitioning patterns: Filter, Composite (2h) • Structural Patterns: Adapter, Façade, Decorator, Iterator, Composite (3h) • Behavioural Patterns: Command, Observer, Strategy, Template Method (3h) **III. Software Architecture Patterns for Enterprise Applications** • Architecture styles and description languages (4h) • Layered Architecture (3) • MVC Pattern (2) • Enterprise Integration Patterns (4)
Laboratory	The practical aspects of the course will involve mostly pattern-oriented solutions for software reuse. Laboratory consists of discussion, problem solving, and presentation of homework solutions. Pre-reading of the lecture notes and class attendance is essential and students are expected to be prepared and to actively participate in class activities. There are about 7 assignments, due two weeks after the student get them. Assignments should be prepared for the next class period. Some may be collected for grading; others will be reviewed in class.
Project	Two projects are assigned during the fall semester. The projects to be undertaken are one-person projects. The first one, the mid-term project, analyse and then design an application by selecting the appropriate design patterns and finally implement it in Java. The second one, the end-term project, should carry out the architectural design of a distributed application. Both projects are a requirement for the competition of the course and will be graded individually. It is the responsibility of the students to document and give full details on their contribution to the project.
Grading	Grading will be for homework assignment and laboratory participation (20%), project elaboration (30%) and two exams (partial exam 25% + final exam 25%). The obtained points will vary depending the design solutions choice, patterns usage, CASE tools usage, documentation quality, and effort.
Professional significance	An auxiliary objective of this course is to give the student a working knowledge of major ideas and concepts of software reusing, largely acknowledged as the major source of productivity gain in software development. The course provides students in Computer Science with skills to create high quality software designs exhibiting improved flexibility, reduced maintenance costs, and with increased understanding of the resulting code.