Universidad Nacional Mayor de San Marcos
School of Computer Science
Syllabus of Course
Academic Period 2018-II

1. **Code and Name:** CS2901. Software Engineering I (Mandatory)
2. **Credits:** 4
3. **Hours of theory and Lab:** 2 HT; 4 HL; (15 weeks)
4. **Professor(s)**

   Meetings after coordination with the professor
5. **Bibliography**

[Lar08] Craig Larman. *Applying UML and Patterns*. Prentice Hall, 2008.

[Pre05] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. 6th. McGraw-Hill, Mar. 2005.

[Som08] Ian Sommerville. *Software Engineering*. 7th. ISBN: 0321210263. Addison Wesley, May 2008.

6. Information about the course

(a) **Brief description about the course** The aim of developing software, except for extremely simple applications, requires the execution of a well-defined development process. Professionals in this area require a high degree of knowledge of the different models and development process, so that they are able to choose the most suitable for each development project. On the other hand, the development of medium and large-scale systems requires the use of pattern and component libraries and the mastery of techniques related to component-based design

(b) **Prerrequisites:**

- CS1103. Objects oriented programming II. (3^{rd} Sem)
- CS2701. Databases I. (4^{th} Sem)

(c) **Type of Course:** Mandatory

(d) **Modality:** Face to face

7. Specific goals of the Course

- Provide the student with a theoretical and practical framework for the development of software under quality standards.
- Familiarize the student with the software modeling and construction processes through the use of CASE tools.
- Students should be able to select architectures and ad-hoc technology platforms for deployment scenarios
- Applying component-based modeling to ensure variables such as quality, cost, and time-to-market in development processes.
- Provide students with best practices for software verification and validation.

8. Contribution to Outcomes

b) An ability to design and conduct experiments, as well as to analyze and interpret data. (**Usage**)

c) An ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability. (**Usage**)

i) An ability to use the techniques, skills, and modern computing tools necessary for computing practice. (**Assessment**)

k) Apply the principles of development and design in the construction of software systems of variable complexity. (**Usage**)

9. Competences (IEEE)

C7. Being able to apply the software engineering principles and technologies to ensure that software implementations are robust, reliable, and appropriate for their intended audience.⇒ **Outcome b,k**

C8. Understanding of what current technologies can and cannot accomplish. ⇒ **Outcome b,c,k**

C12. Understanding the lifecycle implications for the development of all aspects of computer-related systems (including software, hardware, and human computer interface).⇒ **Outcome c,i**

C18. Ability to participate actively and as a member of a team. .⇒ **Outcome k**

CS1. Model and design computer-based systems in a way that demonstrates comprehension of the tradeoff involved in design choices.⇒ **Outcome c**

CS2. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.⇒ **Outcome b,c**

CS4. Deploy appropriate theory, practices, and tools for the specification, design, implementation, and maintenance as well as the evaluation of computer-based systems.⇒ **Outcome b,c,i**

CS5. Specify, design, and implement computer-based systems.⇒ **Outcome b,c,i**

CS10. Deploy effectively the tools used for the construction and documentation of software, with particular emphasis on understanding the whole process involved in using computers to solve practical problems. This should include tools for software control including version control and configuration management.⇒ **Outcome i,k**

10. List of topics

1. Requirements Engineering
2. Software Design
3. Software Construction

11. Methodology and Evaluation
Methodology:

Theory Sessions:
The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

Lab Sessions:
In order to verify their competences, several activities including active learning and roleplay will be developed during lab sessions.

Oral Presentations:
Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

Reading:
Throughout the course different readings are provided, which are evaluated. The average of the notes in the readings is considered as the mark of a qualified practice. The use of the UTEC Online virtual campus allows each student to access the course information, and interact outside the classroom with the teacher and with the other students.
Evaluation System:

12. Content

Unit 1: Requirements Engineering (18)

Competences Expected: C7, C11, CS2

Learning Outcomes	Topics
• List the key components of a use case or similar description of some behavior that is required for a system [Assessment] • Describe how the requirements engineering process supports the elicitation and validation of behavioral requirements [Assessment] • Interpret a given requirements model for a simple software system [Assessment] • Describe the fundamental challenges of and common techniques used for requirements elicitation [Assessment] • List the key components of a data model (eg, class diagrams or ER diagrams) [Assessment] • Identify both functional and non-functional requirements in a given requirements specification for a software system [Assessment] • Conduct a review of a set of software requirements to determine the quality of the requirements with respect to the characteristics of good requirements [Assessment] • Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system [Assessment] • Compare the plan-driven and agile approaches to requirements specification and validation and describe the benefits and risks associated with each [Assessment] • Use a common, non-formal method to model and specify the requirements for a medium-size software system [Assessment] • Translate into natural language a software requirements specification (eg, a software component contract) written in a formal specification language [Assessment] • Create a prototype of a software system to mitigate risk in requirements [Assessment] • Differentiate between forward and backward tracing and explain their roles in the requirements validation process [Assessment]	• Describing functional requirements using, for example, use cases or users stories • Properties of requirements including consistency, validity, completeness, and feasibility • Software requirements elicitation • Describing system data using, for example, class diagrams or entity-relationship diagrams • Non functional requirements and their relationship to software quality • Evaluation and use of requirements specifications • Requirements analysis modeling techniques • Acceptability of certainty / uncertainty considerations regarding software / system behavior • Prototyping • Basic concepts of formal requirements specification • Requirements specification • Requirements validation • Requirements tracing

Readings : [Pre05], [Som08], [Lar08]

Unit 2: Software Design (18)	
Competences Expected: C5, C7, C8, CS10	
Learning Outcomes	**Topics**
• Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation [Familiarity] • Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design [Usage] • Construct models of the design of a simple software system that are appropriate for the paradigm used to design it [Usage] • Within the context of a single design paradigm, describe one or more design patterns that could be applicable to the design of a simple software system [Familiarity] • For a simple system suitable for a given scenario, discuss and select an appropriate design paradigm [Usage] • Create appropriate models for the structure and behavior of software products from their requirements specifications [Usage] • Explain the relationships between the requirements for a software product and its design, using appropriate models [Assessment] • For the design of a simple software system within the context of a single design paradigm, describe the software architecture of that system [Familiarity] • Given a high-level design, identify the software architecture by differentiating among common software architectures such as 3-tier, pipe-and-filter, and client-server [Familiarity] • Investigate the impact of software architectures selection on the design of a simple system [Assessment] • Apply simple examples of patterns in a software design [Usage] • Describe a form of refactoring and discuss when it may be applicable [Familiarity] • Select suitable components for use in the design of a software product [Usage] • Explain how suitable components might need to be adapted for use in the design of a software product [Familiarity] • Design a contract for a typical small software component for use in a given system [Usage] • Discuss and select appropriate software architecture for a simple system suitable for a given scenario [Usage] • Apply models for internal and external qualities in designing software components to achieve an acceptable tradeoff between conflicting quality aspects [Us-	• System design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion , re-use of standard structures • Design Paradigms such as structured design (top-down functional decomposition), object-oriented analysis and design, event driven design, component-level design, data-structured centered, aspect oriented, function oriented, service oriented • Structural and behavioral models of software designs • Design patterns • Relationships between requirements and designs: transformation of models, design of contracts, invariants • Software architecture concepts and standard architectures (e.g. client-server, n-layer, transform centered, pipes-and-filters) • The use of component desing: component selection, design, adaptation and assembly of components, component and patterns, components and objects (for example, building a GUI using a standar widget set) • Refactoring designs using design patterns • Internal design qualities, and models for them: efficiency and performance, redundacy and fault tolerance, traceability of requeriments • Measurement and analysis of design quality • Tradeoffs between different aspects of quality • Application frameworks • Middleware: the object-oriented paradigm within middleware, object request brokers and marshalling, transaction processing monitors, workflow systems • Principles of secure design and coding – Principle of least privilege – Principle of fail-safe defaults – Principle of psychological acceptability

Unit 3: Software Construction (24)

Competences Expected: C4, C5, C7, C8, CS2

Learning Outcomes	Topics
• Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness [Assessment] • Build robust code using exception handling mechanisms [Assessment] • Describe secure coding and defensive coding practices [Assessment] • Select and use a defined coding standard in a small software project [Assessment] • Compare and contrast integration strategies including top-down, bottom-up, and sandwich integration [Assessment] • Describe the process of analyzing and implementing changes to code base developed for a specific project [Assessment] • Describe the process of analyzing and implementing changes to a large existing code base [Assessment] • Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions [Assessment] • Write a software component that performs some non-trivial task and is resilient to input and run-time errors [Assessment]	• Coding practices: techniques, idioms/patterns, mechanisms for building quality programs – Defensive coding practices – Secure coding practices – Using exception handling mechanisms to make programs more robust, fault-tolerant • Coding standards • Integration strategies • Development context: "green field" vs. existing code base – Change impact analysis – Change actualization • Potential security problems in programs – Buffer and other types of overflows – Race conditions – Improper initialization, including choice of privileges – Checking input – Assuming success and correctness – Validating assumptions

Readings : [Pre05], [Som08], [Lar08]