COURSE DATA

DATA SUBJECT

Code: 34840
Name: Software engineering I
Cycle: Undergraduate Studies
ECTS Credits: 6
Academic year: 2026-27

STUDY (S)

Degree	Center	Acad. year	Period
1407 - Degree in Multimedia Engineering	Escola Tècnica Superior d'Enginyeria	3	First quarter

SUBJECT-MATTER

Degree	Subject-matter	Character
1407 - Degree in Multimedia Engineering	Desarrollo del Software Multimedia	COMPULSORY

COORDINATION

MARTINEZ PLUME JAVIER

SUMMARY

The course "Software Engineering" is a core course as part of the Telematics Engineering Degree. The course workload is 6 ECTS and it's offered in the 3rd semester of 3rd year.

The aim of the course is to introduce students in the development of software projects by following a systematic process and relying on tools to improve software quality in production environments.

It will introduce students to the knowledge and use of different methodologies for developing information systems.

We seek to provide sufficient knowledge of the software process, so that students, using the Unified Process, will be able to capture requirements, analyze, design, implement, test and deploy software projects in a concrete and accuracy way.

In regard to the practical part, in this course students will be able to implement the knowledge acquired in the theoretical part using UML modeling language and Java programming language.

The main objective of this course is to introduce students to the development of software projects from requirements analysis to implementation and verification of the product by the customer, relating to the following points:

- Understand the origin and meaning of the term "Software Engineering", its historical development and current challenges (with attention to the sociocultural context of development), and be aware of the ethical and professional responsibility of a Software Engineer.
- Become aware of the importance of always performing the analysis and design of the problem, as prior tasks to implementation in a programming language.
- Be aware of the need of modeling and abstraction in software development.
- Understand the concept of software development method and its main classifications.
- Distinguish the concepts of diagram and model.
- Know the main UML diagrams: use cases, classes, packages, objects, interaction (sequence and communication), states and activities, and be able to apply them in order to model a medium sized project.
- Given an application of medium size, be able to address the requirements analysis focused on use cases, the conceptual or domain modeling, and the analysis of collaborations between objects with appropriate allocation of responsibilities, specially taking into account technological details.
- Understand and apply design techniques within the framework of an iterative process.
- Choose the best option between different data conceptual designs, justifying and arguing the decision.
- Know and apply basic design patterns for building software and evaluate its role as a way of reuse of experience.

Use software tools that allow the creation of different UML diagrams.

PREVIOUS KNOWLEDGE

RELATIONSHIP TO OTHER SUBJECTS OF THE SAME DEGREE

There are no specified enrollment restrictions with other subjects of the curriculum.

OTHER REQUIREMENTS

Without prerequisites for enrollment, it is recommended to have completed the following courses / fields:
Programming

COMPETENCES / LEARNING OUTCOMES

1405 -

B4 - Have basic skills in the use and programming of computers, operating systems, databases and computer software for use in engineering.

B5- Know the structure, organisation, operation and interconnection of computer systems, the fundamentals of their programming and their application to solve engineering problems.

I9- Know and apply the principles, methodologies and life cycles of software engineering.

MM21 - Communicate effectively, both in writing and verbally, knowledge, procedures, results and ideas related to ICT and specifically to multimedia, and know their socioeconomic impact.

MM23 - Make proper use of theories, procedures and tools in the professional development of multimedia engineering in a real context (specification, design, implementation, deployment and evaluation of multimedia systems solutions).

MM24 - Be able to design, develop, evaluate and ensure the accessibility, ergonomics, usability and security of multimedia systems, services and applications and of the information that these manage.

MM26 - Be able to conceive, develop and maintain multimedia systems, services and applications using the methods of software engineering as a tool for quality assurance, according to the knowledge acquired as described in the specific competences.

MM28 - Be able to solve problems with initiative, decision-making and creativity and to communicate and transmit the knowledge, abilities and skills of a multimedia engineer.

MM3 - Be able to implement methodologies, technologies, processes and tools for the professional development of multimedia products in a real context of use by applying the appropriate solutions for each environment.

MM5 - Know how to apply the theoretical and practical resources to deal with a multimedia application as a whole.

DESCRIPTION OF CONTENTS

Skills to be acquired:
Understand what is software engineering and its need
Know and understand the fundamental concepts that comprise the basic terminology of software engineering
Understanding the relationships between the concepts of software process, software lifecycle and software methodology
Knowing the characteristics and explain the advantages and disadvantages of different software process models
Know the main types of software methodologies
Know the basic features of the Unified Process software development
Understand what is software modeling and its benefits
Recognize UML as a standard language to build software
Contents:
1.1 Overview of Software Engineering
1.2 Basics of Software Engineering
1.3 Models and Processes
1.3.1 Introduction to Software Process

1. Introduction to Software Development Process UML

Skills to be acquired:
Understand what is software engineering and its need
Know and understand the fundamental concepts that comprise the basic terminology of software engineering
Understanding the relationships between the concepts of software process, software lifecycle and software methodology
Knowing the characteristics and explain the advantages and disadvantages of different software process models
Know the main types of software methodologies
Know the basic features of the Unified Process software development
1.3.2 Software Process Models
1.3.3 Software Development Methodologies
1.4 The Unified Modeling Language UML 2.0
1.4.1 Structural and Dynamic Modeling
1.4.2 UML Views
1.5 Unified Process of OO Software Development
1.5.1 Characteristics
1.5.2 Phases
1.5.3 Activities and Artifacts

Skills to be acquired:
Understand the value of acquiring and managing requirements and their influence on the success of a project
Understand what requirements are and the complexity of requirements extraction
Learn the requirements activities
Identify the different types of requirements and be able to discern between them
Learn about diverse elicitation techniques to capture system requirements
Understand what is the Requirements Document
Know the IEEE / ANSI 830-199 for SRS
Develop a SRS document for medium size systems
Learn the different elements and diagrams that UML provides to represent Use Cases
Represent Functional Requirements with Use Cases
Accomplish detailed specification of Use Cases
Contents:
2.1. requirements
2.1.1 Definition and characteristics of the Requirements
2.1.2 Types of Requirements
2.1.3 Requirements Activities
2.1.4 Techniques for obtaining Requirements
2.1.5 Software Requirements Specification
2.1.6 Exercises on requirements
2.2 Prototype
2.3 Use Cases
2.3.1 Introduction.
2.3.2 Elements: Actors, Subjects, Events and Scenarios

2. Planning Phase and Specification

Skills to be acquired:
Understand the value of acquiring and managing requirements and their influence on the success of a project
Understand what requirements are and the complexity of requirements extraction
Learn the requirements activities
Identify the different types of requirements and be able to discern between them
Learn about diverse elicitation techniques to capture system requirements
Understand what is the Requirements Document
Know the IEEE / ANSI 830-199 for SRS
Develop a SRS document for medium size systems
Learn the different elements and diagrams that UML provides to represent Use Cases
Represent Functional Requirements with Use Cases
Accomplish detailed specification of Use Cases
2.3.3 Use Case Specification
2.3.4 Relations: performance, generalization, extension, including
2.3.5 Use Case Diagrams
2.3.6 Standard errors and Recommendations
2.3.7 Exercises on use cases.

Skills to be acquired:
Know the steps required to accomplish the analysis phase in the first cycle of development and the artifacts to be generated
Be able to develop the Data Dictionary
Be able to abstract the relevant concepts to develop a Conceptual Model
Develop using Class Diagrams the conceptual model of a system
Identify system events in Use Cases descriptions to extract System Operations
Develop System Sequence Diagrams for Use Cases starting from System Operations
Develop Contracts for System Operations
Represent complex behaviors using State and Activity Diagrams
Contents:
3.1 Conceptual Model
3.2 Class Diagram
3.2.1 Classifiers: Types and Properties
3.2.2 Relations dependency, generalization, association, realization
3.3 Diagram of objects
3.3.1. instances
3.4 Exercises class and object diagrams
3.5 System Sequence Diagrams
3.5.1 Interactions
3.5.2 Lifelines
3.5.3 Posts
3.6 Contracts
3.7 Sequence diagrams and contracts Exercises
3.8 Behavior Diagrams complex and State / Activity
3.8.1 Activity Elements

3. Analysis phase

Skills to be acquired:
Know the steps required to accomplish the analysis phase in the first cycle of development and the artifacts to be generated
Be able to develop the Data Dictionary
Be able to abstract the relevant concepts to develop a Conceptual Model
Develop using Class Diagrams the conceptual model of a system
Identify system events in Use Cases descriptions to extract System Operations
Develop System Sequence Diagrams for Use Cases starting from System Operations
Develop Contracts for System Operations
Represent complex behaviors using State and Activity Diagrams
Contents:
3.1 Conceptual Model
3.2 Class Diagram
3.2.1 Classifiers: Types and Properties3.8.2 Activity Diagrams
3.8.3 State Machine Elements
3.8.4 State Diagrams
3.9 Activity Diagrams and State Exercises

4. Design phase

Skills to be acquired:
Know the steps required to accomplish the design phase in the first cycle of development and the artifacts to be generated
Understand the concept of responsibility
Understand and know how to apply a set of patterns when deciding responsibilities assignment to classes
Be able to develop interaction diagrams for each system operation following its contract
Develop the Design Class Diagram from the Conceptual Model
Contents:
4.1 Model of Design
4.1.1 Responsibilities
4.2 Design Class Diagram
4.3 Interaction Diagrams
4.3.1 Operation sequence diagram
4.3.2 Communication diagram
4.4 interaction diagrams Exercises
4.5 Patterns for the allocation of responsibilities.
4.5.1 GRASP Patterns
4.5.2 Patterns GoF
4.6 Example: Design of TPV

Skills to be acquired:
Understand the concepts of layers, packages and partitions and how to use in organizing the system architecture

5. System Architecture

Skills to be acquired:
Represent packages and their relationships in Package Diagrams
Choose the architecture to be used and model it using Packages Diagrams
Knowing and applying other patterns
Contents:
5.1 Architecture and UML multilayer
5.1.1 Layers and partitions
5.1.2 Packages
5.1.3 Diagram Packages
5.2 Patterns of connection between packages
5.2.1 GRASP Patterns
5.2.2 Patterns GoF

6. Implementation Phase

Skills to be acquired:
Learn prior decisions before implementing
Know the types of transformation from model space to code space
Transform design artifacts into code
Detect models modification need for system optimization
Contents:
6.1 Prior Decisions
6.2 Types of transformation
6.2.1 Model Transformations
6.2.2 Code Transformations
6.2.3 Model to Code Transformations: direct Engineering
6.2.4 Code to Model Transformations: reverse Engineering
6.3 Direct Engineering
6.3.1 Mapping Classes
6.3.2 Mapping Relations
6.3.3 Mapping Inheritance
6.3.4 Methods Creation
6.3.5 Mapping Contracts

Skills to be acquired:
Understand and differentiate the key issues related to software testing
Understand the need for testing as an essential part of software system development
Distinguish the different testing levels according on the purpose
Learn different software testing techniques
Contents:
7.1 Basics: Errors, Defects, Failures, Test Cases
7.2 Verification and Validation
7.2.1 Software Inspections
7.2.2 Software Testing

7. Testing

Skills to be acquired:
Understand and differentiate the key issues related to software testing
Understand the need for testing as an essential part of software system development
Distinguish the different testing levels according on the purpose
Learn different software testing techniques
7.2.3 Debugging
7.3 Software Testing Levels
7.3.1 Unit Testing
7.3.2 Integration Testing
7.3.3 Acceptance Testing
7.3.4 System Tests
7.4 Software Testing Techniques
7.4.1 Black-Box Testing
7.4.2 White-Box Testing
7.5 Test Plan

WORKLOAD

PRESENCIAL ACTIVITIES

Activity	Hours
Theory	30,00
Laboratory	20,00
Classroom practices	10,00
Total hours	**60,00**

NON PRESENCIAL ACTIVITIES

Activity	Hours
Attendance at other activities	0,00
Individual or group project	4,00
Independent study and work	4,00
Preparation of lessons	73,00
Preparation for assessment activities	0,00
Resolution of case studies	9,00
Total hours	**90,00**

TEACHING METHODOLOGY

Teaching activities will be conducted in accordance with the following distribution:

- **Theoretical activities.**

The lectures will present the course contents providing a global vision, a detailed analysis of the key concepts and encouraging the student participation.

- **Practical activities.**

The practical activities complement the theoretical classes and allow the students to put into practice the contents and improve the understanding of the course concepts. They include the following types of classroom activities:

- Solving problems in class.
- Regular discussion of exercises and problems that the students have previously tried to work out.
- Workshops and seminars in computer lab.
- Group work for project planning and software development and generation of group dynamics.
- Support tutorial sessions (individualized).

To carry out these activities, the general group will be subdivided into smaller subgroups (20 students maximum) according to need.

- **Personal work.**

Preparation of classes and exams (study). This is done individually and tries to promote autonomous work habits.

- **Teamwork in small groups.**

It will be carried out by small groups of students (3-4). It consists of work to be done out of the class timetable in form of exercises and problems. This work tries to improve the teamwork and leadership skills. They include the following types of activities:

- Group work research and collecting information on basic concepts of software engineering, software life cycle, agile methodologies, software process models, UML history.
- Presentation of group work (in Castilian).
- Group work for project planning and software development and generation of group dynamics.
- Software development projects, which documentation must be submitted in Castilian and English.
- Presentation of the software project.
- Support tutorial sessions (groups).

During the course the e-learning platform (Aula Virtual) of the University of Valencia will be used to support the teaching activities. This platform allows the access to the course materials used in the classes as well as additional documents, solved problems and exercises.

EVALUATION

Students can choose between two different assessments:

- Continuous assessment system
- Single assessment system.

Continuous assessment system

This will be the method recommended to students. This system allows regular assessment of students' participation, their exploit of training activities and their participation in the learning process.

Following aspects will be valued:

- **Theory sessions:** involvement will be assessed, taking into account regular attendance to planned classroom activities, delivery of the exercises and participation in their resolution, including work on units 1 and 2 (Theory_M).
- **Problems sessions:** involvement will be assessed, taking into account regular attendance to planned classroom activities, delivery of the exercises, participation in solving exercises during classes and active participation in the forums (Problem_M).
- **Laboratory sessions:** involvement will be assessed, taking into account regular attendance to planned classroom activities and delivery of the proposed exercises (Laboratory _M).
- **Individual examination:** consisting of one or several exams, or knowledge test, that will include both theoretical and practical questions and problems (Exam_M).

To apply such an assessment a class attendance rate above 75% will be required. This percentage will applied separately to each block. That is, students should attend to more than 75% of theory sessions, more than 75% of the practice sessions and more than 75% of the laboratory sessions.

Only works submitted before the date stipulated by the teacher will be considered. This includes exercises in class (theory and practical sessions), laboratory exercises, the work of the first two units and the software project.

A minimum mark of 4 (out of 10) for each part (*Theory_M, Problem_M, Laboratory _M* and *Exam_M*) is required to obtain a final average mark.

The final mark will be obtained applying the following formula:

Final Mark=20% Continuous_M +30%Laboratory_M +50% Exam_M

Continuous_M =50% Theory_M +50% Problem_M

Single assessment system

This method applies to any student who cannot attend classes regularly due to a reasonable and supported by professor cause or who already fail the first session continuous assessment.

Theory sessions mark (*Theory_M*), problems sessions mark (*Problem_M*) and Laboratory sessions mark (*Laboratory _M*) will not be recovered by any other activity.

In both methods, the evaluation will be conducted according to the Regulation of Qualifications of the University of Valencia. At the time of writing this guide, the current legislation is approved by the Governing Council of the UVEG of January 27, 2004, adjusted as provided for that purpose by the Royal Decrees 1044/2003 and 1125 / 2003. It states basically that the marks will be numerical from 0 to 10 with a decimal expression and must be added the qualitative rating scale for the following:

From 0 to 4,9: "Fail"

From 5 to 6,9: "Pass"

From 7 to 8,9: "Very good"

From 9 to 10: "Outstanding" or "with Distinction

Copying or plagiarism of any activity that is part of the evaluation will result in the impossibility of passing the course, and the student will then be subject to the appropriate disciplinary procedures indicated in the ACTION PROTOCOL FOR FRAUDULENT PRACTICES AT THE UNIVERSITY OF VALENCIA (ACGUV 123/2020).

REFERENCES

- Apuntes de la asignatura

- [Roger S. Pressman (2009)] Software Engineering: A Practitioner's Approach, 7th Edition (Mc Graw Hill)

- [I. Sommerville (2011)] Software Engineering, 9th Edition (Addison-Wesley)

- [S. Sánchez Alonso, M. A. Sicilia Urbán, D. Rodríguez García (2011)] Ingeniería de software: un enfoque desde la guía SWEBOK (Garceta)

- [Grady Booch, James Rumbaugh, Ivar Jacobson (2005)] The Unified Modeling Language User Guide (2nd Rev. Edition) (Addison-Wesley)

- [C. Larman (2004)] Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd (Edition Prentice Hall)

- [Bernd Bruegge, Allen H. Dutoit] Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition (Edition Prentice Hall)

- [Kenneth E. Kendall, Julie E Kendall (2010)] Systems Analysis and Design, 8th Edition (Prentice Hall)

- [Michael R. Blaha, James R Rumbaugh (2005)] Object-Oriented Modeling and Design with UML (2nd Edition) (Prentice Hall)

- [A. Weitzenfeld (2004)] Ingeniería de software orientada a objetos con UML, Java e Internet (Thomson)

- [Robert C. Martin (2003)] UML for Java programmers (Prentice Hall)