Software Engineering

1. **Course number and name:** 020GLOES5/020SOEES5 Software Engineering

2. **Credits and contact hours:** 4 ECTS credits, 2x1:15 contact hours

3. **Name of course coordinator:** Rima Kilany

4. **Instructional materials:** course slides (Moodle); lab sessions (Moodle)

   References:
   - UML2 et les designs patterns (Craig Larman – Pearson Education)
   - Refactoring Java/J2EE (EYROLLES)
   - UML Distilled (Martin Fowler)
   - UML2 par la pratique (R.Roques - EYROLLES)
   - Test Logiciel en pratique (John Watkins)
   - Génie Logiciel (David Gustafon – SCHAUM'S)
   - Refactoring: Improving the Design of Existing Code
   - by Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William
   - Opdyke, don Roberts
   - https://www.coursera.org/specializations/product-management
   - https://app.pluralsight.com/library/courses/devops-big-picture/table-of-contents
   - Software Engineering Practices and Methods, Dr. Laurie Williams

5. **Specific course information**
   a. **Catalog description:**
      This course describes the problems related to programming in the Large vs programming in the Small, at the level of cost, quality, functionalities and time management. It explains the methodologies related to the project development life cycle according to sturdy traditional approaches, such as CMM, TSP, PSP, RUP as well as according to agile methodologies such as, XP and Scrum (concepts, roles and ceremonies) as well as the waterfall and iterative lifecycles. It details elicitation techniques and software Requirement Specification writing rules and templates, as well as it describes many specification tools used for functional and non-functional requirements analysis. It explains the DRY, KISS and SOLID principles mainly its advanced object-oriented design concepts (OCP, LSP, etc…), and covers UML diagrams for OO modeling. It also explains the CRC Card design method adopted by the eXtreme Programming methodology. It demonstrates the need for continuous refactoring and explains refactoring techniques at a chirurgical, tactical and strategic level. It also describes the process to follow in order to succeed in refactoring, starting by configuring and using configuration/source code management tools like Git/GitHub, as well as testing and bug management software, then, by evaluating the quantitative and qualitative code quality in order to find eligible refactoring candidates and finally by executing and validating the refactoring step.

This course describes the testing pyramid and details unit/integration/functional and non-functional testing, while stressing on the need for Test Driven development using JUnit. It compares methods that can be used to estimate the cost of a software. It explains UI/UX to-do and not-do basics by studying the different cases of standalone, and web applications focusing on accessibility issues. Finally, it introduces DevOps principles and raises students' awareness about SAAS development and the value of IT automation.

b. **Prerequisites:** None

c. **Required** for CCE Software Engineering Option students; **Selected Elective** for students in the CCE Artificial Intelligence and Telecommunication Networks Options.

6. **Educational objectives for the course**
   a. **Specific outcomes of instruction:**
      - Understand the issues related to Programming in the Large Vs Programming in the Small.
      - Choose a suitable methodology/life cycle and personalize a process in order to adapt to the nature of the solution to implement, and succeed in respecting cost, time, quality and required functionalities constraints.
      - Understand SCRUM concepts, roles and ceremonies and apply those concepts.
      - Understand the DevOps cycle and the principles of IT automation in the context of a SAAS (Software As A service) development.
      - Write suitable Software Requirement Specification (functional and non-functional requirements, conceptual model), after the analysis and identification of client needs by choosing the most appropriate elicitation and specification techniques.
      - Design a solution at a high level by choosing the most appropriate software architecture, and at a detailed level in an object-oriented context and use UML as a modeling language.
      - Use the appropriate methodology and tools to refactor and/or maintain a project solution, in order to fix or prevent bugs or to enhance software quality, such as source code management tools, testing and bug management tools.
      - Analyze software quality, quantitatively (static code analysis, code metrics) and qualitatively (search for architectural anti-patterns, code inspection) as a step in software refactoring process.
      - Apply Test Driven Development, using JUnit, and understand integration, functional and non-functional testing.
      - Understand and compare different approaches for estimating the cost of a software development.
      - Evaluate UI/UX design for standalone, and web applications and understand accessibility design issues.

b. PI addressed by the course:

PI	1.1	2.1	2.2	2.3	2.4	2.5	4.1	4.2	7.1	7.1
Covered	x	x	x	x	x	x	x	x	x	x
Assessed	x	x	x	x	x	x	x	x		

7. Brief list of topics to be covered

- Introduction to software engineering: requirements and constraints of programming in the Large (1 Lecture)
- Activities of the software development process (1 Lecture)
- Software Life cycles and methodologies: Traditional vs Agile (3 Lectures)
- Agile Methodologies eXtreme Programming and SCRUM (concepts, roles and ceremonies) (3 Lectures)
- Elicitation: artefacts and techniques (1 Lecture)
- Software Requirement Specifications (functional requirements, non-functional requirements, conceptual models): tools, techniques, writing rules and templates (2 Lectures)
- Object Oriented Software Design- software architectures - CRC Card design sessions- SOLID, DRY, KISS design principles Advanced Object-oriented design concepts- Implementation best practices and code conventions) (2 Lectures)
- Refactoring: levels, process, source code management, testing and bugs management tools (2 Lectures)
- Software quality evaluation: Qualitative analysis (code inspection, architecture review, anti-patterns) (1 Lecture)
- Software quality evaluation: Quantitative analysis (code metrics) (1 Lecture)
- Configuration Management - Versionning tools (1 Lecture)
- Refactoring: a practical example (1 Lecture)
- Software Testing: Difficulties, classification, testing pyramid (2 Lectures)
- Testing at all the life cycle levels (Unit, Integration, functional, and non-functional) (1 Lecture)
- Estimation of a software development cost (1 Lecture)
- UI design: Evolution, Elements (1 Lecture)
- UI design: Web, and Accessibility (1 Lecture)
- UI/UX design: User experience evaluation (1 Lecture)
- UML History, Modeling with UML (1 Lecture)
- UML Views – Functional View (1 Lecture)
- UML: Static view (2 Lectures)
- UML: Dynamic view (2 Lectures)
- Lab: UML hands-on for a real life use-case (2 Lectures)
- Software As Service – IT Automation – DevOps the big picture (1 Lecture)