National University of Engineering (UNI)
School of Artificial Intelligence
Syllabus 2027-I

1. COURSE

CS291. Software Engineering I (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS291. Software Engineering I
2.2 Semester	:	5th Semester.
2.3 Credits	:	4
2.4 Hours	:	2 HT; 4 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 Prerrequisites	:	• CS113. Objects-oriented programming II. (3rd Sem) • CS271. Data Management. (4th Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

Software engineering is a collaborative discipline that applies structured processes to build high-quality systems. This first course focuses on the foundational aspects of the software lifecycle, from initial requirements and human-centric design to systematic testing. By integrating teamwork principles and rigorous development fundamentals, students will learn to create software that is both technically sound and user-oriented.

5. GOALS

- Understand and apply various software process models and configuration management.
- Master requirements elicitation and modeling using standardized notations.
- Develop effective teamwork and collaboration skills for software development.
- Design robust software architectures based on engineering principles.
- Implement systematic testing strategies and understand software quality.

6. COMPETENCES

AG-C03) Individual and Team Work: Performs effectively as an individual and as a member or leader in diverse teams. (Usage)

AG-C09) Design and Development of Solutions: Designs, implements, and evaluates solutions for complex computing problems. (Usage)

7. TOPICS

Unit 1: Version Control and CI/CD (12 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Software configuration management and version control: Software Development Practices – Configuration in version control, reproducible builds/configuration. – Version control branching strategies. Development branches vs release branches. Trunk-based development. – Merging/rebasing strategies, when relevant. • Release management. • Testing tools including static and dynamic analysis tools. Software Development Practices, Information Flow and Non-Interference, Injection and Input Validation, Memory Safety and Types, Malware Analysis and Advanced Security • Software process automation: – Build systems - the value of fast, hermetic, reproducible builds, compare/contrast approaches to building a project. – Continuous Integration (CI) - the use of automation and automated tests to do preliminary validation that the current head/trunk revision builds and passes (basic) tests. – Continuous Deployment (CD) - the use of automation to automatically release every change that passes the automated tests to the production environment, ensuring frequent and reliable deliveries. – Dependency management - updating external/upstream dependencies, package management, SemVer. • Collaborative review workflow as a quality gate – Mandatory pull-request review before merging – Code ownership via CODEOWNERS-style rules – Conventional commit message conventions – Branch-protection rules enforcing checks before merge • Dependency and supply-chain vulnerability scanning integrated into CI pipelines	• Describe the difference between centralized and distributed software configuration management [Describe] • Describe how version control can be used to help manage software release management [Describe] • Identify configuration items and use a source code control tool in a small team-based project [Analyze] • Describe how available static and dynamic test tools can be integrated into the software development environment [Describe] • Understand the use of CI/CD systems as a ground-truth for the state of the team's shared code (build and test success) [Explain] • Apply a pull-request review workflow with code-ownership rules and branch protection to enforce quality gates on a team project [Apply] • Use a dependency/supply-chain vulnerability scanner as part of a CI pipeline to detect vulnerable dependencies [Use]
Readings : [**Sommerville2015**], [**Pressman2019**]	

Unit 2: Requirements Engineering (14 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Describe functional requirements using, for example, use cases or user stories. – Using at least one method of documenting and structuring functional requirements. – Understanding how the method supports design and implementation. – Strengths and weaknesses of using a specific approach. • Properties of requirements including consistency, validity, completeness, and feasibility. • Requirements elicitation. – Sources of requirements, for example, users, administrators, or support personnel. – Methods of requirement gathering, for example, surveys, interviews, or behavioral analysis. • Non-functional requirements, for example, security, usability, or performance, also called as Quality Attributes. Sustainable Design and Pervasive Computing, Environmental Footprint of Computing Systems, Systemic Effects and Social Context • Risk identification and management, including ethical considerations surrounding the proposed product. Ethics Codes and Legal Accountability, Professional Responsibility and Public Policy, Professional Values, Nature and Continuing Development, Ethical Dissent and Cultural Context, Workplace Ethics • Communicating and/or formalizing requirement specifications.	• Compare different methods of eliciting requirements along multiple axes [Compare] • Identify differences between two methods of describing functional requirements (e.g., customer interviews, user studies) and the situations where each would be preferred [Analyze] • Identify which behaviors are required, allowed, or barred from a given set of requirements and a list of candidate behaviors [Analyze] • Collect a set of requirements for a simple software system [Analyze] • Identify areas of a software system that need to be changed, given a description of the system and a set of new requirements to be implemented [Analyze] • Identify the functional and non-functional requirements in a set of requirements [Analyze]
Readings : [**Sommerville2015**], [**Wiegers2013**]	

Unit 3: Team Collaboration and Dynamics (14 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Effective communication, including oral and written, as well as formal (email, docs, comments, presentations) and informal (team chat, meetings). Technical and Professional Communication, Team Communication and Culture • Common causes of team conflict, and approaches for conflict resolution. • Cooperative programming: – Pair programming or Swarming – Code review – Collaboration through version control • Roles and responsibilities in a software team: Ethics Codes and Legal Accountability, Professional Responsibility and Public Policy, Professional Values, Nature and Continuing Development, Ethical Dissent and Cultural Context, Workplace Ethics – Advantages of teamwork – Risks and complexity of such collaboration • Team processes - responsibilities for tasks, effort estimation, meeting structure, work schedule	• Follow effective team communication practices [Analyze] • Articulate the sources of, hazards of, and potential benefits of team conflict - especially focusing on the value of disagreeing about ideas or proposals without insulting people [Articulate] • Facilitate a conflict-resolution and problem-solving strategy in a team setting [Analyze] • Collaborate effectively in cooperative development/programming [Analyze] • Propose and delegate necessary roles and responsibilities in a software development team [Propose] • Compose and follow an agenda for a team meeting [Compose] • Facilitate through involvement in a team project, the central elements of team building, establishing healthy team culture, and team management including creating and executing a team work plan [Create]
Readings : [**Wigdor2011**], [**Johnson2013**]	

Unit 4: Principles and Software Architecture (12 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**

• System design principles. System Reliability
  – Levels of abstraction (e.g., architectural design and detailed design)
  – Separation of concerns
  – Information hiding
  – Coupling and cohesion
• Software architecture. System Reliability
  – Design paradigms
  ∗ Top-down functional decomposition/layered design
  ∗ Data-oriented architecture
  ∗ Object-oriented analysis and design
  ∗ Event-driven design
  – Standard architectures (e.g., client-server and microservice architectures including REST discussions, n-layer, pipes-and-filters, Model View Controller)
  – Identifying component boundaries and dependencies
• Programming in the large vs programming in the small. System Reliability
• Code smells and other indications of code quality, distinct from correctness. Security Design and Controls Engineering, Threat Analysis and Security Engineering, Trusted Computing and Privacy Engineering
	• Identify the standard software architecture of a given high-level design [Analyze]
	• Select and use an appropriate design paradigm to design a simple software system and explain how system design principles have been applied in this design [Evaluate]
	• Adapt a flawed system design to better follow principles such as separation of concerns or information hiding [Create]
	• Identify the dependencies among a set of software components in an architectural design [Analyze]

Readings : [**Gamma1994**], [**Pressman2019**]	

Unit 5: Verification and validation Concepts (12 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Verification and validation concepts – Verification: Are we building the thing right? – Validation: Did we build the right thing? • Why testing matters: Does the component remain functional as the code evolves? • Testing objectives – Usability – Reliability – Conformance to specification – Performance – Security	• Explain why testing is important [Explain] • Distinguish between program validation and verification [Distinguish] • Describe different objectives of testing [Describe]
Readings : [**Sommerville2015**], [**Pressman2019**]	

Unit 6: Testing Tools and Analysis (4 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Verification and validation tooling and automation – Static analysis – Code coverage – Fuzzing – Dynamic analysis and fault containment (sanitizers, etc.) – Fault logging and fault tracking • Test planning and generation – Fault estimation and testing termination including defect seeding – Use of random and pseudo random numbers in testing • Testing asynchronous, parallel, and concurrent systems • Verification and validation of non-code artifacts (documentation, training materials) • Behavior-driven development (BDD) – Expressing acceptance criteria as Gherkin Given/When/Then scenarios – Executing BDD scenarios as automated end-to-end tests • Mutation testing and static code-quality analysis – Mutation testing to assess test-suite effectiveness – Static detection of code smells and excessive cyclomatic complexity	• Describe and compare different tools for verification and validation [Describe] • Automate the use of different tools in a small software project [Design] • Explain how and when random numbers should be used in testing [Explain] • Describe approaches for fault estimation [Describe] • Estimate the number of faults in a small software application based on fault density and fault seeding [Estimate] • Describe techniques and issues with testing asynchronous, concurrent, and parallel software [Describe] • Create a test plan for a medium-size code segment which contains asynchronous, concurrent, and/or parallel code, including a test matrix and generation of test data and inputs [Create] • Describe techniques for the verification and validation of non-code artifacts [Describe] • Write Gherkin Given/When/Then scenarios for a feature's acceptance criteria and execute them as automated end-to-end tests [Write] • Analyze a test suite's mutation score and refactor tests to kill surviving mutants [Analyze]
Readings : [**Sommerville2015**], [**Pressman2019**]	

Unit 7: Performance Testing and Benchmarking (4 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**Learning Outcomes**
• Performance testing and benchmarking – Throughput and latency – Degradation under load (stress testing, FIFO vs LIFO handling of requests) – Speedup and scaling ∗ Amdahl's law ∗ Gustafson's law ∗ Soft and weak scaling – Identifying and measuring figures of merits – Common performance bottlenecks ∗ Compute-bound ∗ Memory-bandwidth bound ∗ Latency-bound – Statistical methods and best practices for benchmarking ∗ Estimation of uncertainty ∗ Confidence intervals – Analysis and presentation (graphs, etc.) – Timing techniques	• Describe throughput and latency and provide examples of each [Describe] • Explain speedup and the different forms of scaling and how they are computed [Explain] • Describe common performance bottlenecks [Describe] • Describe statistical methods and best practices for benchmarking software [Describe] • Explain techniques for and challenges with measuring time when constructing a benchmark [Explain] • Identify the figures of merit, construct and run a benchmark, and statistically analyze and visualize the results for a small software project [Analyze]
Readings : [**Bondi2015**], [**Gregg2020**]	

8. WORKPLAN

8.1 Methodology

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

8.2 Theory Sessions

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

8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

9. EVALUATION SYSTEM

********* EVALUATION MISSING ********

10. BASIC BIBLIOGRAPHY