



National University of Engineering (UNI)
School of Computer Science
Syllabus 2026-II

1. COURSE

CS395. Software Quality (Elective)

2. GENERAL INFORMATION

2.1 Course	: CS395. Software Quality
2.2 Semester	: 8 th Semester.
2.3 Credits	: 3
2.4 Hours	: 2 HT; 2 HP;
2.5 Duration of the period	: 16 weeks
2.6 Type of course	: Elective
2.7 Learning modality	: Face to face
2.8 Prerequisites	: CS292. Software Engineering II. (7 th Sem)

3. PROFESSORS

Lecturer

- Ernesto Cuadros-Vargas <ecuadrosv@uni.edu.pe>
 - PhD in Computer Science, ICMC-USP, Brazil, 2004.
 - MSc in Computer Science, ICMC-USP, Brazil, 1998.

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

Delivering software that is merely "functional" is no longer sufficient for industry: modern organizations demand systems that are reliable, secure, maintainable, and built through disciplined, auditable engineering processes. This course teaches software quality as a first-class, hands-on engineering discipline practiced through the same tools and workflows used by professional teams on GitHub: branch protection, mandatory code review, automated testing at every level, continuous integration/continuous delivery (CI/CD), static analysis, and production observability.

Students progress from foundational quality models through advanced collaborative-development workflows, automated testing strategies (unit, integration, end-to-end, and behavior-driven), CI/CD pipelines, static analysis and refactoring, and production-quality concerns such as supply-chain security, performance, and reliability engineering. The course closes with an emphasis on DevOps culture – the shared team practices and values (Definition of Done, blameless postmortems, continuous learning) that sustain quality at scale.

5. GOALS

- Evaluate software quality using an internationally recognized quality-characteristics model and reason about the cost of poor quality and technical debt.
- Apply a professional, review-gated Git/GitHub workflow, including branch protection, CODEOWNERS, and conventional commits.
- Design and implement unit, integration, end-to-end, and behavior-driven test suites, including test-driven development.
- Build and operate CI/CD pipelines that enforce automated quality gates, dependency scanning, and semantic versioning.
- Apply static analysis, mutation testing, and refactoring techniques to identify and remove code smells and technical debt.
- Analyze production-quality concerns, including performance testing and reliability/observability practices.
- Adopt DevOps culture practices – Definition of Done, blameless postmortems, and continuous team learning – to sustain quality.

6. COMPETENCES

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

AG-C11) Use of Tools: Applies modern computing tools in problem solving. (Assessment)

AG-C12) Uses systemic approaches to select, develop, apply, integrate, and manage secure computing technologies to achieve user objectives. (Usage)

7. TOPICS

Unit 1: Software Design Quality and Evaluation (4 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	Learning Outcomes
• Data design Data Modeling – Data structures – Storage systems • Requirement traceability – Understanding which requirements are satisfied by a design • Design modeling, for instance with class diagrams, entity relationship diagrams, or sequence diagrams • Measurement and analysis of design quality • Principles of secure design and coding Security Design and Controls Engineering, Threat Analysis and Security Engineering, Trusted Computing and Privacy Engineering – Principle of least privilege – Principle of fail-safe defaults – Principle of psychological acceptability • Evaluating design tradeoffs (e.g., efficiency vs reliability, security vs usability) • ISO/IEC 25010 quality characteristics model and cost of poor quality – Quality characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, portability – Cost of poor quality and technical debt as a business/economic concern, not just a technical one	• Design a set of data structures to implement a provided API surface [Design] • Identify which requirements are satisfied by a provided software design [Analyze] • Translate a natural language software design into class diagrams [Translate] • Adapt a flawed system design to better follow the principles of least privilege and fail-safe defaults [Create] • Contrast two software designs across different qualities, such as efficiency or usability [Contrast] • Evaluate a software system against the ISO/IEC 25010 quality characteristics and estimate the business cost of poor quality/technical debt [Evaluate]
Readings : [Int11], [Som15]	

Unit 2: Version Control and CI/CD (12 hours)	
Competences Expected: AG-C11,AG-C12	
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 : [Kim+21]	

Unit 3: Test Planning and Types (8 hours)	
Competences Expected: AG-C09	
Topics	Learning Outcomes
• Test kinds – Unit – Integration – Validation – System • Stylistic differences between tests and production code: DAMP vs DRY - more duplication is warranted in test code. • Test planning and generation – Test case generation, from formal models, specifications, etc. – Test coverage * Test matrices * Code coverage - how much of the code is tested? * Environment coverage - how many hardware architectures, operating systems, browsers, etc. are tested? – Test data and inputs	• Compare and contrast the different types and levels of testing (regression, unit, integration, systems, and acceptance) [Compare] • Describe techniques for creating a test plan and generating test cases [Describe] • Create a test plan for a medium-size code segment which includes a test matrix and generation of test data and inputs [Create] • Implement a test plan for a medium-size code segment [Implement]
Readings : [Som15]	

Unit 4: Advanced Testing Practices (4 hours)	
Competences Expected: AG-C11	
Topics	Learning Outcomes
• Test development – Test-driven development – Object oriented testing, mocking, and dependency injection – Opaque-box (previously, black-box) and transparent-box (previously, white-box) testing techniques – Test tooling, including code coverage, static analysis, and fuzzing • Verification and validation in the development cycle – Code reviews – Test automation, including automation of tooling – Pre-commit and post-commit testing – Tradeoffs between test coverage and throughput/latency of testing – Defect tracking and prioritization: reproducibility of reported defects • Domain specific verification and validation challenges – Performance testing and benchmarking – Asynchrony, parallelism, and concurrency – Safety-critical – Numeric	• Identify the fundamental principles of test-driven development methods and explain the role of automated testing in these methods [Analyze] • Discuss issues involving the testing of object-oriented software [Debate] • Describe mocking and dependency injection and their application [Describe] • Undertake, as part of a team activity, a code review of a medium-size code segment [Evaluate] • Describe the role that tools can play in the validation of software [Describe] • Automate the testing in a small software project [Design] • Explain the roles, pros, and cons of pre-commit and post-commit testing [Explain] • Discuss the tradeoffs between test coverage and test throughput/latency and how this can impact verification [Debate] • Use a defect tracking tool to manage software defects in a small software project [Use] • Discuss the limitations of testing in certain domains [Debate]
Readings : [Bec02]	

Unit 5: Testing Tools and Analysis (8 hours)	
Competences Expected: AG-C11	
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 : [WH12]	

Unit 6: API Compatibility and Versioning (4 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
• Hyrum's Law/The Law of Implicit Interfaces • Backward compatibility – Compatibility is not a property of a single entity, it's a property of a relationship. – Backward compatibility needs to be evaluated in terms of provider + consumer(s) or with a well-specified model of what forms of compatibility a provider aspires to/promises. • Versioning – Semantic Versioning (SemVer) – Trunk-based development	• Identify both explicit and implicit behavior of an interface and identify potential risks from Hyrum's Law [Analyze] • Identify changes that can be broadly considered "backward compatible," potentially with explicit statements about what usage is or is not supported [Analyze] • Evaluate whether a proposed change is sufficiently safe given the versioning methodology in use for a given project [Evaluate]
Readings : [Som15]	

Unit 7: Refactoring Techniques (4 hours)	
Competences Expected: AG-C09,AG-C12	
Topics	Learning Outcomes
• Refactoring – Standard refactoring patterns (rename, inline, outline, etc.) – Use of refactoring tools in IDE – Application of static-analysis tools (to identify code in need of refactoring, generate changes, etc.) – Value of refactoring as a remedy for technical debt • "Large Scale" Refactoring - techniques when a refactoring change is too large to commit safely (large projects), or when it is impossible to synchronize change between provider + all consumers (multiple repositories, consumers with private code). – Express both old and new APIs so that they can co-exist. – Minimize the size of behavior changes. – Why these techniques are required, (e.g., "API consumers I can see" vs "consumers I can't see").	• Consider inputs from static analysis tools and/or Software Design principles to identify code in need of refactoring [Analyze] • Refactor the implementation of an interface to improve design, clarity, etc. with minimal/zero impact on existing users [Create] • Plan a complex multi-step refactoring to change default behavior of an API safely [Plan]
Readings : [Fow17]	

Unit 8: Performance Testing and Benchmarking (4 hours)	
Competences Expected:	
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 : [Bon15], [Gre20]	

Unit 9: Reliability Engineering (4 hours)	
Competences Expected: AG-C11,AG-C12	
Topics	Learning Outcomes
• Software reliability models • Software fault tolerance techniques and models – Contextual differences in fault tolerance (e.g., crashing a flight critical system is strongly avoided, crashing a data processing system before corrupt data is written to storage is highly valuable) • Software reliability engineering practices - including reviews, testing, practical model checking • Identification of dependent and independent failure domains, and their impact on system reliability • Measurement-based analysis of software reliability - telemetry, monitoring and alerting, dashboards, release qualification metrics, etc.	• Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system [Demonstrate] • Identify methods that will lead to the realization of a software architecture that achieves a specified level of reliability [Analyze] • Identify ways to apply redundancy to achieve fault tolerance [Analyze] • Identify single-point-of-failure (SPF) dependencies in a system design [Analyze]
Readings : [Kim+21], [FHK18]	

Unit 10: Large-Scale Construction and Process (4 hours)	
Competences Expected: AG-C12	
Topics	Learning Outcomes
• Larger-scale testing – Test doubles (stubs, mocks, fakes) – Dependency injection • Work sequencing, including dependency identification, milestones, and risk retirement – Dependency identification: Identifying the dependencies between different tasks – Milestones: A collection of tasks that serve as a marker of progress when completed. Ideally, the milestone encompasses a useful unit of functionality. – Risk retirement: Identifying what elements of a project are risky and prioritizing completing tasks that address those risks. • Potential security problems in programs Information Flow and Non-Interference, Injection and Input Validation, Memory Safety and Types, Malware Analysis and Advanced Security – Buffer and other types of overflows – Race conditions – Improper initialization, including choice of privileges – Input validation • Documentation (autogenerated) • Development context: "green field" vs existing code base – Change impact analysis – Change actualization • Release management • DevOps practices – Definition of Done as a shared quality bar for completed work – Blameless postmortems for learning from incidents/failures – Team quality culture: shared ownership, feedback, and continuous learning	• Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions [Apply] • Write a software component that performs some non-trivial task and is resilient to input and run-time errors [Write]
Readings : [FHK18]	

Unit 11: Capstone: End-to-End Quality Pipeline Demonstration (8 hours)	
Competences Expected: AG-C09	
Topics	Learning Outcomes
- Integration of all course practices into a single live repository: branch protection, mandatory PR review, automated test suites (unit, integration, E2E/BDD), CI/CD pipeline with quality gates, dependency scanning, static analysis, and observability. - Live repository review: demonstration and defense of the team's quality pipeline in front of peers/instructor. - Retrospective and blameless postmortem of the project's quality journey.	- Integrate version control, automated testing, CI/CD, static analysis, and observability into a single, end-to-end quality pipeline for a real repository. [Usage] - Defend the design and effectiveness of a team's quality pipeline through a live repository review. [Usage] - Conduct a blameless postmortem to identify process improvements for future projects. [Assessment]
Readings : [FHK18]	

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

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 our virtual campus allows each student to access the course information, and interact outside the classroom with the teacher and with the other students.

The final grade NF depends on the following grades:

- PE : Partial Exam.
- FE : Final Exam.
- P : Permanent Evaluation (Average of practical work and graded assignments).

The final grade NF is calculated as follows:

$$NF = \frac{PE + FE + P}{3}$$

10. BASIC BIBLIOGRAPHY

- [Bec02] Kent Beck. *Test-Driven Development: By Example*. Addison-Wesley, 2002.
- [Int11] International Organization for Standardization. *ISO/IEC 25010:2011 – Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models*. ISO/IEC. ISO/IEC, 2011.
- [WH12] Matt Wynne and Aslak Hellesøy. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2012.
- [Bon15] André B. Bondi. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*. Upper Saddle River, NJ: Addison-Wesley, 2015.
- [Som15] Ian Sommerville. *Software Engineering*. 10th. Pearson, 2015.
- [Fow17] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. 2nd. Addison-Wesley, 2017.
- [FHK18] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018.
- [Gre20] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. 2nd. Boston, MA: Addison-Wesley Professional, 2020.
- [Kim+21] Gene Kim et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. 2nd. IT Revolution Press, 2021.