MODULE DESCRIPTION FORM

نموذج وصف المادة الدراسية

Module Information معلومات المادة الدراسية					
Module Title	**Software Engineering**			**Module Delivery**	
Module Type	**Core Learning Activity**			☒ **Theory** ☐**Lecture** ☐ **Lab** ☐ **Tutorial** ☒ **Practical** ☐ **Seminar**	
Module Code	**SOEN321**				
ECTS Credits	6.00				
SWL (hr/sem)	**150**				
Module Level	2		**Semester of Delivery**	1	
Administering Department	Type Dept. Code		**College**	Type College Code	
Module Leader	**Dr. Omar Janeh**		**e-mail**	120049@uotechnology.edu.iq	
Module Leader's Acad. Title	**Lecturer**		**Module Leader's Qualification**	Ph.D.	
Module Tutor	**None**		**e-mail**	**None**	
Peer Reviewer Name	1. Dr. Riadh Jabar 2. Azhar Malik		**e-mail**	Dhari.a.mahmood@uotechnology.edu.iq 120016@uotechnology.edu.iq	
Scientific Committee Approval Date	**1 / 4 /2026**		**Version Number**	1.0	

Relation with other Modules العلاقة مع المواد الدراسية الأخرى			
Prerequisite module	None	**Semester**	
Co-requisites module	None	**Semester**	

Module Aims, Learning Outcomes and Indicative Contents أهداف المادة الدراسية ونتائج التعلم والمحتويات الإرشادية	
Module Objectives أهداف المادة الدراسية	1. Understand SDLC Principles: Develop a strong foundation in Software Development Life Cycle (SDLC) models, including traditional and agile methodologies. 2. Master Software Design: Learn to analyze requirements, design software architectures, and create effective object-oriented models using UML. 3. Develop Practical Coding Skills: Gain hands-on experience with Python programming, including debugging, testing, and implementing design patterns. 4. Apply DevOps Practices: Understand and apply DevOps principles for continuous integration, delivery, deployment, and monitoring. 5. Enhance Software Quality: Explore software testing techniques, quality assurance practices, and tools for ensuring reliability, maintainability, and security.
Module Learning Outcomes مخرجات التعلم للمادة الدراسية	1. Understand the scope, importance, and key principles of software engineering, including its methodologies and process models. 2. Explain and differentiate between SDLC phases (planning, analysis, design, implementation, testing, and maintenance). 3. Learn how to elicit, document, and validate functional and non-functional software requirements. 4. Apply design principles such as modularity, abstraction, and reusability in software system design. 5. Analyze and use software architectural patterns like MVC, layered architecture, and client-server models. 6. Utilize version control tools (e.g., Git) for collaborative and iterative software development. 7. Develop software applications in Python, adhering to coding standards, and following best practices for maintainability. 8. Understand and apply different types of software testing (unit testing, integration testing, system testing, and acceptance testing). 9. Implement continuous integration/continuous deployment (CI/CD) pipelines using tools like Jenkins, Docker, and Kubernetes. 10. Work within Agile development frameworks, including creating user stories, sprints, and conducting retrospectives. 11. Apply software quality metrics to evaluate the performance, reliability, and maintainability of a system. 12. Integrate security considerations into the software development process, focusing on secure coding and threat modeling. 13. Address ethical, legal, and professional challenges in software engineering, including intellectual property and data privacy. 14. Plan and manage software projects, create documentation, and prepare for project handoff or maintenance. 15. Demonstrate learned skills by presenting a completed project that encompasses requirements analysis, design, implementation, testing, and

	deployment.
Indicative Contents المحتويات الإرشادية	**Module 1: Fundamentals of Software Engineering (25 Hours)** **Topics:** 1. Definition, importance, engineering principles. 2. Overview of SDLC models (Waterfall, Agile, Spiral). 3. Introduction to software processes and case studies. **Activities:** 1. Case studies on SDLC. 2. Group discussions on engineering principles. **Module 2: Requirements Analysis and Software Design (30 Hours)** **Topics:** 1. Eliciting requirements, stakeholder analysis. 2. Functional and non-functional specifications. 3. UML diagrams, modular design, SOLID principles. 4. Design patterns (Singleton, Factory, Observer). **Activities:** 1. Create UML diagrams for a case project. 2. Design and review modular components. **Module 3: Software Development and Programming (30 Hours)** **Topics:** 1. Python programming best practices. 2. Secure coding and debugging. 3. Implementing modular programming. **Activities:** 1. Hands-on coding with Python (group and individual). 2. Debugging and code reviews. 3. Build a basic application with modular code. **Module 4: Software Testing and Quality Assurance (30 Hours)** **Topics:** 1. Testing strategies (unit, integration, system). 2. Automated testing tools (Pytest, Selenium basics). 3. Metrics for software quality and maintainability.

	Activities: 1. Writing and running test cases in Python. 2. Evaluating software quality using metrics. **Module 5: Agile Practices and DevOps (35 Hours)** **Topics:** 1. Agile principles, Scrum, and sprint planning. 2. CI/CD pipeline concepts. 3. Tools: Git, Docker, Kubernetes basics. **Activities:** 1. Setting up and managing a Git repository. 2. Building a CI/CD pipeline. 3. Running Docker containers for an app. **Module 6: Software Project Management and Capstone Project (40 Hours)** **Topics:** 1. Project planning, risk management, documentation. 2. Integration of all learned concepts into a final project. **Activities:** 1. Develop a project proposal. 2. Work on team-based software engineering projects. 3. Present and deploy the final software solution.

Learning and Teaching Strategies استراتيجيات التعلم والتعليم	
Strategies	1. **Lectures**: Instructors will deliver in-class lectures to introduce and explain key networking concepts, architectures, and protocols. These presentations will cover theoretical foundations and practical applications. 2. **Interactive Discussions**: Students will be encouraged to participate in discussions that foster critical thinking and problem-solving skills. These discussions will revolve around case studies, hypothetical scenarios, and current events in networking. 3. **Hands-on Laboratory Work**: Practical lab sessions will allow students to apply theoretical knowledge by working with networking hardware and simulation software. This includes setting up networks, configuring routers

	and switches, and analyzing network traffic with tools like Wireshark and Cisco Packet tracer. 4. **Group Projects**: Students will collaborate on projects that involve designing and implementing network solutions for simulated environments. This promotes teamwork and the practical application of learned concepts, including network planning, security measures, and troubleshooting. 5. **Simulations and Virtual Labs**: Utilizing advanced simulation tools and virtual lab environments to provide students with hands-on experience, especially when physical resources or access to actual networking equipment is limited. 6. **Use of Visuals and Multimedia**: Integration of visual aids such as diagrams, flowcharts, and multimedia content to enhance understanding of complex network structures and data flow mechanisms. 7. **Assessment and Feedback**: Regular assessments through quizzes, tests, and exams to gauge students' understanding and mastery of course content. Feedback will be provided systematically to guide students' learning processes and adjustments. 8. **Practice and Revision Sessions**: Dedicated sessions for practice and revision will be available to reinforce learning, address students' questions, and prepare them adequately for assessments.

Student Workload (SWL)

الحمل الدراسي للطالب محسوب لـ ١٥ اسبوعا

Structured SWL (h/sem) الحمل الدراسي المنتظم للطالب خلال الفصل	**48**	**Structured SWL (h/w)** الحمل الدراسي المنتظم للطالب أسبوعيا	**3**
Unstructured SWL (h/sem) الحمل الدراسي غير المنتظم للطالب خلال الفصل	**77**	**Unstructured SWL (h/w)** الحمل الدراسي غير المنتظم للطالب أسبوعيا	**3**
Total SWL (h/sem) الحمل الدراسي الكلي للطالب خلال الفصل	**125**		

Module Evaluation

تقييم المادة الدراسية

	Time/Number	Weight (Marks)	Week Due	Relevant Learning Outcome

Formative assessment	**Quizzes**	2	10% (10)	5 and 10	LO #1- #5 and #6 - #11
	Assignments	2	10% (10)	3 and 12	LO #2 and #5, #7, #11
	Projects / Lab.	1	10% (10)	Continuous	All
	Report	1	10% (10)	13	LO #5 - #12
Summative assessment	**Midterm Exam**	2hr	10% (10)	8	LO #1 - #10
	Final Exam	3hr	50% (50)	16	All
Total assessment			100% (100 Marks)		

Delivery Plan (Weekly Syllabus)

المنهاج الاسبوعي النظري

	Material Covered
Week 1	**Introduction to Software Engineering** 1. History and importance of software engineering. 2. Overview of the SDLC and process models (Waterfall, Agile, DevOps). 3. Ethical issues and software engineering code of conduct (ACM/IEEE). 4. Case Study: Analyze the failure of a large software project.
Week 2	**Requirements Engineering** 1. Techniques for requirements elicitation (interviews, surveys, focus groups). 2. Writing Software Requirement Specifications (SRS). 3. UML use case diagrams and activity diagrams. 4. Hands-On: Write an SRS document and draw UML diagrams for a case study.
Week 3	**Software Design and Architecture** 1. Principles of good design: modularity, abstraction, and encapsulation. 2. Architectural patterns (MVC, client-server, microservices). 3. UML class and sequence diagrams. 4. Hands-On: Design a small application using Python with UML diagrams.
Week 4	**Version Control and Collaboration** 1. Introduction to Git and GitHub.

	2. Branching strategies, pull requests, and resolving conflicts. 3. Collaborative workflows in team projects. 4. Hands-On: Use Git for version control and manage a team repository.
Week 5	**Programming Practices and Code Quality** 1. Clean code principles (SOLID, DRY, KISS). 2. Refactoring and code reviews. 3. Documentation and API design. 4. Hands-On: Refactor and document a provided Python codebase.
Week 6	**Agile and Scrum in Software Development** 1. Agile manifesto and principles. 2. Scrum framework: roles, events, and artifacts. 3. Creating product backlogs and writing user stories. 4. Hands-On: Use a tool like Jira/Trello to manage a mini-sprint.
Week 7	**Testing and Quality Assurance** 1. Importance of testing: unit, integration, and acceptance testing. 2. Python testing frameworks: unittest, pytest. 3. Test-driven development (TDD). 4. Hands-On: Write unit tests for Python functions and implement TDD.
Week 8	**DevOps and CI/CD** 1. Introduction to DevOps principles and pipelines. 2. CI/CD tools: GitHub Actions, Jenkins, or GitLab CI/CD. 3. Automating tests and deployments. 4. Hands-On: Create a basic CI/CD pipeline for automated testing.
Week 9	**Containerization and Deployment** 1. Introduction to Docker and Kubernetes. 2. Deploying applications to cloud platforms (AWS, Azure, or Google Cloud). 3. Monitoring and scaling applications. 4. Hands-On: Containerize a Python application using Docker and deploy it to a cloud platform.
Week 10	**Software Maintenance and Security** 1. Types of maintenance: corrective, adaptive, perfective, and preventive.

	2. Basics of secure coding practices. 3. Identifying and mitigating vulnerabilities. 4. Hands-On: Implement input validation and error handling in a Python project.
Week 11	**Emerging Trends in Software Engineering** 1. AI and machine learning in software development. 2. Blockchain for software security. 3. Low-code and no-code development platforms. 4. Case Study: Discuss emerging trends in the software industry.
Week 12-14	**Final Project Development** Students will work in teams to apply SDLC processes to a real-world software project: 1. Week 12: Requirements analysis and system design. 2. Week 13: Implementation, testing, and CI/CD setup. 3. Week 14: Deployment and final touches. Deliverables: Functional software product, SRS document, design diagrams, test cases, and deployment pipelines.
Week 15	**Final Project Presentations and Review** 1. Teams present their projects to peers and instructors. 2. Peer reviews and instructor feedback. 3. Wrap-up discussion on lessons learned and career pathways in software engineering.
Delivery Plan (Weekly Practical Syllabus) المنهاج الاسبوعي للدروس العملية	
	Material Covered
Week 1	**Introduction to Software Engineering** **Practical Goals:** Familiarize students with SDLC models and tools for project management. **Tasks:** 1. Set up project management tools (Trello/Jira). 2. Create a simple project plan for a software project using Waterfall and Agile models. 3. Discuss ethical considerations in software engineering using a real-world case study.
Week 2	**Requirements Engineering** **Practical Goals:** Gather and document software requirements. **Tasks:**

	1. Conduct a mock interview with a client to gather requirements. 2. Write an SRS document for a simple application. 3. Create UML use case and activity diagrams using Lucidchart or similar tools.
Week 3	**Software Design and Architecture** **Practical Goals:** Design software architecture and modular components. **Tasks:** 1. Create UML class and sequence diagrams for an e-commerce platform. 2. Design a basic modular architecture using MVC. 3. Implement a basic Python program to demonstrate modularity.
Week 4	**Version Control and Collaboration** **Practical Goals:** Master Git for version control and team collaboration. **Tasks:** 1. Initialize a Git repository and push code to GitHub. 2. Practice branching, merging, and resolving conflicts. 3. Collaborate on a small project using pull requests.
Week 5	**Programming Practices and Code Quality** **Practical Goals:** Apply clean code principles and improve code quality. **Tasks:** 1. Refactor a poorly written Python program to follow clean code principles (SOLID, DRY). 2. Conduct peer code reviews within teams. 3. Document a Python library using docstrings and README files.
Week 6	**Agile and Scrum in Software Development** **Practical Goals:** Use Agile and Scrum methodologies in team projects. **Tasks:** 1. Create a product backlog and sprint plan using Jira/Trello.

	2. Write user stories and acceptance criteria for a simple project. 3. Conduct a mock sprint planning meeting.
Week 7	Testing and Quality Assurance **Practical Goals:** Implement and automate testing. **Tasks:** 1. Write unit tests for a Python application using unittest and pytest. 2. Conduct manual testing for functionality and usability. 3. Practice test-driven development (TDD) by writing tests before coding.
Week 8	**DevOps and CI/CD** **Practical Goals:** Automate testing and deployment pipelines. **Tasks:** 1. Create a basic CI/CD pipeline using GitHub Actions or Jenkins. 2. Automate testing and deployment of a Python program. 3. Explore logs and results of automated builds.
Week 9	**Containerization and Deployment** **Practical Goals:** Learn Docker and deploy applications. **Tasks:** 1. Install Docker and containerize a Python application. 2. Deploy the containerized application locally. 3. Deploy the application to a cloud platform like AWS or Google Cloud.
Week 10	**Software Maintenance and Security** **Practical Goals:** Maintain and secure software. **Tasks:** 1. Perform code maintenance by fixing bugs and adding new features. 2. Identify and fix security vulnerabilities in a Python program (e.g., input validation). 3. Conduct a security audit of an existing project.
Week 11	**Emerging Trends in Software Engineering** **Practical Goals:** Explore modern software development trends. **Tasks:** 1. Implement a simple AI/ML model using Python (e.g., scikit-learn).

	2. Create a small blockchain application for secure transactions. 3. Experiment with low-code platforms like AppGyver.
Week 12	**Final Project Development (Requirements and Design)** **Practical Goals:** Start a team project, focusing on requirements and design. **Tasks:** 1. Gather requirements and create an SRS document for the final project. 2. Design the system architecture and create UML diagrams. 3. Prepare a detailed project timeline and milestones.
Week 13	**Final Project Development (Implementation and Testing)** **Practical Goals:** Develop and test the final project. **Tasks:** 1. Implement the core features of the project. 2. Write unit and integration tests. 3. Set up a CI/CD pipeline for automated testing.
Week 14	**Final Project Development (Deployment and Refinement)** **Practical Goals:** Finalize and deploy the project. **Tasks:** 1. Deploy the project to a cloud platform. 2. Conduct final bug fixes and refinements. 3. Prepare a presentation for the project.
Week 15	**Final Project Presentation and Review** **Practical Goals:** Present the project and reflect on the learning process. **Tasks:** 1. Present the project to the class and instructors. 2. Conduct peer reviews of other projects. 3. Write a reflective report on lessons learned.

Learning and Teaching Resources

مصادر التعلم والتدريس

	Text	Available in the Library?
Required Texts	1. Sommerville, Ian. Software Engineering (10th Edition). 2. Martin, Robert C. Clean Code: A Handbook of Agile Software Craftsmanship.	Yes

	3. Kim, Gene et al. The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win.	
Recommended Texts		
Websites	1. GitHub Learning Lab: https://lab.github.com 2. Docker Documentation: https://docs.docker.com 3. Python Official Docs: https://docs.python.org	

Grading Scheme مخطط الدرجات				
Group	**Grade**	التقدير	**Marks %**	**Definition**
Success Group (50 - 100)	**A -** Excellent	امتياز	90 - 100	Outstanding Performance
	B - Very Good	جيد جدا	80 - 89	Above average with some errors
	C - Good	جيد	70 - 79	Sound work with notable errors
	D - Satisfactory	متوسط	60 - 69	Fair but with major shortcomings
	E - Sufficient	مقبول	50 - 59	Work meets minimum criteria
Fail Group (0 – 49)	**FX –** Fail	راسب (قيد المعالجة)	(45-49)	More work required but credit awarded
	F – Fail	راسب	(0-44)	Considerable amount of work required

Note: Marks Decimal places above or below 0.5 will be rounded to the higher or lower full mark (for example a mark of 54.5 will be rounded to 55, whereas a mark of 54.4 will be rounded to 54. The University has a policy NOT to condone "near-pass fails" so the only adjustment to marks awarded by the original marker(s) will be the automatic rounding outlined above.