MODULE DESCRIPTION FORM

نموذج وصف المادة الدراسية

Module Information معلومات المادة الدراسية					
Module Title	**Software Engineering**			**Module Delivery**	
Module Type	**Core**			☒**Theory** ☒**Lecture** ☒ **Lab** ☐**Tutorial** ☐**Practical** ☐**Seminar**	
Module Code	**CPE 206**				
ECTS Credits	**6**				
SWL (hr/sem)	**150**				
Module Level	2		**Semester of Delivery**	3	
Administering Department	Computer Eng.		**College**	College of Engineering	
Module Leader	Dr. Khalid Jamal Jadaa		**e-mail**	Khalid.jamal.jadaa@uodiyala.edu.iq	
Module Leader's Acad. Title			**Module Leader's Qualification**		
Module Tutor	Name (if available)		**e-mail**	E-mail	
Peer Reviewer Name	Ghassan Khazal Ali		**e-mail**	Ghassan_khazal@uodiyala.edu.iq	
Scientific Committee Approval Date	02/06/2024		**Version Number**	1.0	

Relation with other Modules العلاقة مع المواد الدراسية الأخرى			
Prerequisite module		**Semester**	
Co-requisites module		**Semester**	

Module Aims, Learning Outcomes and Indicative Contents أهداف المادة الدراسية ونتائج التعلم والمحتويات الإرشادية	
Module Objectives أهداف المادة الدراسية	Upon completion of this course, the student will be able to: 1. List and describe the fundamental phases of the Software Development Lifecycle (SDLC) 2. Define and describe fundamental software engineering terminology and coding practices 3. Explore/explain relationships between software engineering and other engineering disciplines (Systems Engineering, Electrical and Computer Engineering, Industrial Engineering) 4. Modify/build a software program that introduces students to software development tools / environments 5. Troubleshoot and debug changes made to an existing software program 6. Develop an original Python software program, learning basic Python language syntax 7. Build a foundation for academic success in the Software Engineering degree program
Module Learning Outcomes مخرجات التعلم للمادة الدراسية	1. Describe basic software development and computing fundamentals that make up the Software Development Lifecycle. 2. Explore relationships between software engineering and other engineering disciplines (Systems Engineering, Electrical and Computer Engineering, Industrial Engineering, and Computer Science) 3. Experiment with and use traditional software development process and testing tools, such as configuration management, interpreters/compilers and debuggers. 4. Analyze the functionality and performance of software application programs 5. Compare and contrast how diverse software applications produce solutions to meet specific objectives/needs in a variety of fields including, but not limited to public health, safety, global, cultural, social, environmental, and economic applications 6. Demonstrate and communicate software engineering principles effectively through written reports and/or verbal presentations. 7. Summarize both ethical and professional responsibilities of a software engineer. 8. Build a foundation for academic success in the Software Engineering degree program.
Indicative Contents المحتويات الإرشادية	**1. Introduction to Software Engineering:** • **What is Software Engineering?** Definition, goals, principles, and importance in today's world. • **The Software Development Lifecycle (SDLC):** Various models (Waterfall, Agile, Spiral, etc.), their advantages and disadvantages. • **Software Engineering Ethics and Professionalism:** Code of ethics, social impact of software, and responsible software development. **2. Requirements Engineering:**

- **Elicitation and Analysis:** Gathering, analyzing, and documenting user needs, functional and non-functional requirements.
- **Requirements Specification and Validation:** Writing clear and unambiguous requirements documents, techniques for validation and verification.
- **Requirements Management:** Tracking, controlling, and managing changes to requirements throughout the development process.

3. Software Design:

- **Design Principles:** Principles like modularity, abstraction, cohesion, coupling, etc.
- **Architectural Design:** High-level design, choosing the right architecture for the project (layered, client-server, etc.).
- **Detailed Design:** Designing individual components, data structures, algorithms, and interfaces.
- **Design Patterns:** Common reusable solutions for recurring design problems.

4. Software Construction:

- **Coding Standards and Best Practices:** Adhering to coding conventions, writing clean and maintainable code.
- **Testing:** Unit testing, integration testing, system testing, user acceptance testing, and different types of testing methods.
- **Debugging and Troubleshooting:** Finding and fixing bugs, techniques for debugging and tracing errors.

5. Software Deployment and Maintenance:

- **Deployment Strategies:** Release planning, deployment methods, and managing different environments.
- **Software Maintenance:** Corrective, adaptive, perfective maintenance, and managing software evolution.
- **Software Configuration Management:** Version control systems, managing source code and other project artifacts.
- **Software Quality Assurance:** Quality metrics, software quality assurance methods, and ensuring the delivered software meets requirements.

6. Software Project Management:

- **Project Planning and Estimation:** Defining project scope, creating timelines, and estimating effort and resources.
- **Risk Management:** Identifying, assessing, and mitigating risks throughout the project lifecycle.
- **Team Management:** Communication, collaboration, and leadership skills for managing software development teams.
- **Software Metrics and Measurement:** Tracking progress, measuring performance, and improving development processes.

7. Advanced Software Engineering Topics (Optional):

- **Software Architecture and Design Patterns:** In-depth study of architectural patterns and design patterns.
- **Software Security:** Secure coding practices, vulnerabilities, and security testing.

	• **Software Reliability and Fault Tolerance:** Ensuring software robustness and resilience to errors.
	• **Software Engineering for Cloud and Mobile Applications:** Developing software for modern platforms.
	• **Agile Software Development:** Deep dive into Agile methodologies like Scrum and Kanban.
	• **DevOps and Continuous Integration/Continuous Delivery (CI/CD):** Automating software development and deployment processes.
	8. Case Studies and Practical Projects:
	• Applying the principles of software engineering through real-world case studies and hands-on projects.
	• This helps students understand the practical application of the concepts learned in theory.
	9. Tools and Technologies:
	• Introduction to popular software engineering tools and technologies like:
	  ○ **Version Control Systems:** Git, SVN
	  ○ **Integrated Development Environments (IDEs):** Eclipse, Visual Studio, IntelliJ IDEA
	  ○ **Project Management Tools:** Jira, Trello
	  ○ **Testing Tools:** JUnit, Selenium
	  ○ **Modeling Tools:** UML modeling tools
	Assessment:
	Assessment can include:
	• Assignments and projects
	• Quizzes and exams
	• Class participation
	• Presentation and reports

Learning and Teaching Strategies استراتيجيات التعلم والتعليم	

Strategies	**1. Project-Based Learning (PBL) at the Core:**
	• **The Principle:** Students learn best by doing. PBL allows them to apply theoretical concepts to real-world problems.
	• **Implementation:**
	  ○ **Small, Iterative Projects:** Start with smaller projects that build on each other.
	  ○ **Teamwork:** Encourage collaboration, communication, and conflict resolution within teams.
	  ○ **Real-World Relevance:** Connect projects to actual industry challenges or real-world issues.

- **Feedback and Iteration:** Regular feedback and opportunities to refine designs and code based on feedback.

2. Agile Learning Methodology:

- **The Principle:** Agile principles focus on flexibility, collaboration, and continuous improvement. This aligns well with modern software development practices.
- **Implementation:**
  - **Short Iterations:** Break down projects into short sprints (1-2 weeks) to allow for flexibility and rapid feedback.
  - **Daily Stand-ups:** Brief daily team meetings to discuss progress, roadblocks, and upcoming tasks.
  - **Frequent Demonstrations:** Regular show-and-tell sessions where teams present their work and receive feedback.
  - **Retrospectives:** Reflecting on each sprint to identify areas for improvement in processes and collaboration.

3. Hands-on Coding and Tools:

- **The Principle:** Theoretical concepts become tangible when students write code and use industry-standard tools.
- **Implementation:**
  - **IDEs and Version Control:** Ensure students are proficient with an IDE (e.g., Visual Studio, Eclipse) and a version control system (e.g., Git).
  - **Open Source Projects:** Encourage contributing to or studying open-source projects to gain experience.
  - **Coding Challenges and Exercises:** Regularly assign coding exercises to reinforce concepts and develop problem-solving skills.

4. Industry Guest Speakers and Mentors:

- **The Principle:** Real-world perspectives and insights from professionals can inspire and guide students.
- **Implementation:**
  - **Guest Lectures:** Invite software engineers or project managers to share their experiences and challenges.
  - **Mentorship Programs:** Connect students with industry professionals for guidance and career advice.
  - **Case Studies:** Present real-world case studies of software projects, highlighting challenges, decisions, and outcomes.

5. Emphasis on Soft Skills:

- **The Principle:** Software engineering is not just about technical skills; effective communication, teamwork, and problem-solving are crucial.
- **Implementation:**
  - **Communication Exercises:** Emphasize clear and concise communication in project reports, presentations, and team interactions.
  - **Teamwork Dynamics:** Encourage students to work in diverse teams and learn to navigate different personalities and work styles.
  - **Problem-Solving Techniques:** Develop analytical and critical thinking skills to tackle complex software engineering problems.

	6. Balancing Theory and Practice: • **The Principle:** A strong theoretical foundation is essential, but it must be grounded in practical application. • **Implementation:** ○ **Interactive Lectures:** Integrate coding demonstrations, interactive exercises, and case studies into lectures. ○ **Project-Driven Learning:** Frame theoretical topics within the context of the ongoing project work. ○ **Assignments with Real-World Applications:** Structure assignments that directly relate to industry practices and software engineering challenges. **7. Continuous Assessment and Feedback:** • **The principle:** Regular feedback and assessment help students identify areas for improvement and track their progress. • **Implementation:** ○ **Frequent Code Reviews:** Peer and instructor code reviews to provide constructive feedback on code quality and design. ○ **Project Milestones:** Regular deadlines and milestones for project deliverables, allowing for ongoing assessment. ○ **Self-Assessment:** Encourage students to reflect on their learning and progress through self-assessment exercises.

Student Workload (SWL)

الحمل الدراسي للطالب محسوب لـ ١٥ أسبوعا

Structured SWL (h/sem) الحمل الدراسي المنتظم للطالب خلال الفصل	**63**	**Structured SWL (h/w)** الحمل الدراسي المنتظم للطالب أسبوعيا	**4.2**
Unstructured SWL (h/sem) الحمل الدراسي غير المنتظم للطالب خلال الفصل	**87**	**Unstructured SWL (h/w)** الحمل الدراسي غير المنتظم للطالب أسبوعيا	**5.8**
Total SWL (h/sem) الحمل الدراسي الكلي للطالب خلال الفصل	**150**		

Module Evaluation

تقييم المادة الدراسية

		Time/Number	Weight (Marks)	Week Due	Relevant Learning Outcome
Formative assessment	**Quizzes**	**2**	**20% (10)**	**6 and 12**	**LO #1 to #4 and #6 to #8**
	Assignments	**2**	**10% (5)**	**4, 7 and 10**	**LO #2, #3, #4, #5 and #7,#8,#9**

	Projects / Lab.	**1**	**10% (10)**		
	Report				
Summative assessment	**Midterm Exam**	**1 hr**	**10% (10)**	**9**	**LO #1 - #7**
	Final Exam	**3 hr**	**50% (50)**	**16**	**All**
Total assessment			**100% (100 Marks)**		

Delivery Plan (Weekly Syllabus) المنهاج الاسبوعي النظري	
	Material Covered
Week 1	Introduction
Week 2-3	Software processes
Week 4	Agile software development
Week 5-6	Requirements engineering
Week 7	System modeling
Week 8	Architectural design
Week 9-10	Design and implementation
Week 11-12	Software testing
Week 12-13	Software evolution
Week 14-15	Project management
Week 16	**Preparatory week before the final Exam**

Delivery Plan (Weekly Lab. Syllabus) المنهاج الاسبوعي للمختبر	
	Material Covered
Week 1	Introduction to Software Engineering Lab - Setting Up Development Environment, Version Control Systems
Week 2-3	Software Requirements Analysis - Use Case Diagrams, Data Flow Diagrams, Entity Relationship Diagrams
Week 4-5	System Design - Class Diagrams, Sequence Diagrams, Activity Diagrams
Week 6-7	Implementation and Coding - Coding techniques, Unit Testing
Week 8-9	Integration and Testing - Integration Testing, Debugging Techniques

Week 10-11	Design Project1 E-binding
Week 12-13	Design Project 2 electronic cash counter

Learning and Teaching Resources
مصادر التعلم والتدريس

	Text	Available in the Library?
Required Texts	1. Software Engineering, 10th Edition Author: by Ian Somerville, pearson , 2016 2. Tony Gaddis, "Starting out with Visual C#.", Fourth edition, Boston, Pearson Inc., 2017.	Yes
Recommended Texts	• ROGER S. PRESSMAN BRUCE R. MAXIM Software Engineering a practitioner's approach, NINTH EDITION, 2019. • Ian Sommerville - Engineering Software Products_ An Introduction to Modern Software Engineering-Pearson (2020) • Salvatore A. Buono, "C# and Game Programming: A Beginner's Guide." Second Edition, Boca Raton, CRC Press Inc., 2019. • Faraz Rasheed, "Programmer's Heaven: C# School.", First Edition, Fuengirola, Synchron Data, 2006.	No
Websites	▪ Any other materials available on the web	

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.