CMP2101 Software Engineering

Period per Week			Contact Hour per Semester	Weighted Total Mark	Weighted Exam Mark	Weighted Continuous Assessment Mark	Credit Units
LH	PH	TH	CH	WTM	WEM	WCM	CU
45	00	30	60	100	40	100	4

Rationale

Software engineering is the discipline concerned with the application of theory, knowledge, and practice to build effectively and efficiently software systems that satisfy the requirements of users and customers. Software engineering is applicable to small, medium, and large-scale systems. It encompasses all phases of the life cycle of a software system. The life cycle includes requirement analysis and specification, design, construction, testing, and operation and maintenance. The development of programs benefits from the concepts and practices derived from software engineering.

Software engineering employs engineering methods, processes, techniques, and measurement. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. Software development, which can involve an individual developer or a team of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment.

Objectives

- To introduce the student to the evolution and scope of Software Engineering as a discipline and as a Profession
- To expose the student to the practical imperatives underpinning software development – requirements analysis, design, implementation, testing, deployment, user training and support, and maintenance

Course Content

1. History and Overview

- Indicate some reasons for studying software engineering
- Highlight some people that influenced or contributed to the area of software engineering
- Indicate some important topic areas such as the software process, requirements, specifications, design, testing, validation, evolution, and project management
- Contrast software engineering with computer engineering
- Mention some examples that would use the software engineering approach

- Indicate the existence of formalized software processes such as the software life cycle
- Explain that requirements and specifications may change slightly as a software project evolves
- Indicate the importance of language selection when doing software design
- Highlight the importance of testing and validation in a software projects
- Explore some additional resources associated with software engineering
- Explain the purpose and role of software engineering in computer engineering

2. Software Processes

- Software life cycle and process models (Waterfalls, Spiral, Extreme programming, Agile, RAD, RUP)
- Process assessment models
- Software process metrics

3. Software Requirements and Specifications

- Requirements elicitation
- Requirements analysis modeling techniques
- Functional and nonfunctional requirements
- Prototyping
- Basic concepts of formal specification techniques
- Software Requirements Specification

4. Software Design

- Fundamental design concepts and principles
- Software architecture (Software quality attributes – Runtime, Business, and Engineering; Architectural views and viewpoints; The technical architecting process; Organisational architecting; Architectural styles)
- Modeling with UML
- Structured design
- Object-oriented analysis and design
- Domain-driven development
- Design for reuse

5. Software Verification and Validation

- Verification (Static and Dynamic)
- Validation planning
- Testing fundamentals, including test plan creation and test case generation
- Black-box and white-box testing techniques
- Unit, integration, validation, and system testing
- Object-oriented testing
- Inspections

6. Software Evolution

- Software maintenance: the different forms of maintenance; the associated disciples and the role and the nature of configuration management and version control
- Impact analysis
- Regression testing
- Associated software support
- Characteristics of maintainable software
- Software re-use in its different forms – their strengths and weaknesses

- Reengineering
- Legacy systems
- Software retirement

7. Software Tools and Environments

- Programming environments
- Requirements analysis and design modeling tools
- Testing tools
- Configuration management tools
- Tools based on databases – their design and development
- Additional possibilities including CASE tools
- Tool integration mechanisms

8. Language Translation

- The range of tools that support software development for the computer engineer; the role of a formal semantics of a language
- Different possibilities regarding language translation: comparison of interpreters and compilers for high-level languages, and silicon compilers for hardware description languages, additional possibilities
- Language translation phases (lexical analysis, parsing, generation phase, optimization); separate compilation or translation - the benefits and the mechanisms; machine-dependent and machine-independent aspects of translation

9. Software Project Management

- Team management: team processes; team organization and decision-making, roles and responsibilities in a software team; role identification and assignment; project tracking; team problem resolution
- Project scheduling
- Software measurement and estimation techniques
- Risk analysis
- Software quality assurance
- Software configuration management
- Project management tools
- Software project documentation

10. Software Fault Tolerance

- Software reliability models
- Software fault-tolerance methods: N-version programming, recovery blocks, rollback and recovery
- Fault tolerance in operating systems and data structures
- Fault tolerance in database and distributed systems

Learning Outcomes

On completion of this course the student will be able to:

- Select, with justification, the software development models most appropriate for the development and maintenance of diverse software products; and explain the role of process maturity models.
- Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system; use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system (e.g., structured analysis or object-oriented-analysis); conduct a review of a

software requirements document using best practices to determine the quality of the document; and translate into natural language a software requirements specification written in a commonly used formal specification language.

- Evaluate the quality of multiple software designs based on key design principles and concepts; using a software requirement specification and a common program design methodology and notation, create and specify the software design for a medium-size software product (e.g., using structured design or object-oriented design); and using appropriate guidelines, conduct the review of a software design.
- Demonstrate the application of the different types and levels of testing (unit, integration, systems, and acceptance) to software products of medium size; undertake, as part of a team activity, an inspection of a medium-size code segment; and describe the role that tools can play in the validation of software.
- Identify the principal issues associated with software evolution and explain their impact on the software life cycle; develop a plan for re-engineering a medium-sized product in response to a change request; discuss the advantages and disadvantages of software reuse; and demonstrate the ability to exploit opportunities for software reuse in a variety of contexts.
- Select, with justification, an appropriate set of tools to support the software development of a range of software products; analyze and evaluate a set of tools in a given area of software development (e.g., management, modeling, or testing); and demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.
- Compare and contrast compiled and interpreted execution models, outlining the relative merits of each; describe the phases of program translation from source code to executable code and the files produced by these phases; explain the differences between machine-dependent and machine-independent translation; and show the manner in which these differences are evident in the translation process.
- Demonstrate through involvement in a team project the central elements of team building and team management; prepare a project plan for a software project that includes estimates of size and effort, a schedule, resource allocation, configuration control, change management, and project risk identification and management; and compare and contrast the different methods and techniques used to assure the quality of a software product.
- Understand the concept of software faults and reliability of software; understand various redundancy methods used to allow software to detect software faults and produce correct results in the presence of software faults; and understand software fault tolerance approaches used in operating systems, database systems, and distributed systems.

Recommended and Reference Books

[1] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2004 ISBN 007301933X, 97800783019338

[2] Ian Sommerville. *Software Engineering*, Pearson/Addison-Wesley, 2004. ISBN 0321210263, 9780321210265

[3] IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems

[4] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications (Revision of IEEE Std 830-1993)

[5] Len Bass, Paul Clements, Rick Kazman, April 11, 2003. Software Architecture in Practice, Second Edition. Addison Wesley. ISBN: 0-321-15495-9

[6] Jeff Garland &Richard Anthony, *Large Scale Software Architecture,* John Wiley and Sons, ISBN 0 470 84849 9

[7] Stephen T. Albin, 2003. *The Art of Software Architecture: Design Methods and Techniques*, John Wiley & Sons, ISBN:0471228869

[8] Bennett, S, 2005. *Object-Oriented Systems Analysis and Design Using UML,* 3rd Edition, McGraw-Hill, ISBN: 0077110005 , ISBN 13: 9780077110000

[9] Donald Yates and Tony Wakefield, 2004. *Systems Analysis and Design,* Second Edition, Prentice Hall, Pearson Education Limited. ISBN 0273 65536 1

[10] IEEE Std 730 – 2002. *IEEE Standard for Software Assurance Plans*

[11] IEEE Std 829- 1998. *IEEE Standard for Software Test Documentation*