Knight Foundation School of Computing and Information Sciences

Course Title: Software Engineering I **Date:** 5/27/2020

Course Number: CEN 4010

Number of Credits: 3

Subject Area: Software Engineering	**Subject Area Coordinator:** Monique Ross **email:** moross@cs.fiu.edu
Catalog Description: Software Process Model, Software Analysis and Specification, Software Design, Software Testing	
Textbook: Ashmore, S., Runyan, K. (2015). Introduction to Agile Methods, ISBN#: 978-0-321-92956-3	
References: Fowler, M. (2004). UML Distilled: A brief guide to the standard object modeling languge, 3rd edition	
Prerequisites Courses: CGS 3095 and COP 3337	
Corequisites Courses: None	

Type: Required for CS Major

Prerequisites Topics:

- Programming
- Data Structures
- Oral and written communication skills

Course Outcomes:

1. Be familiar with the Software Development Life Cycle and software process models
2. Master the techniques to gather and specify the requirements of a medium-size software system
3. Master the techniques to design and implement a medium-size software system using UML
4. Be familiar with software testing techniques
5. Be familiar with system walkthroughs
6. Be familiar with software documentation
7. Be familiar with working in a small software development team
8. Demonstrate the ability to communicate the details of the technical solution through verbal and written modes.

Relationship between Course Outcomes and Program Outcomes

BS in CS: Program Outcomes	Course Outcomes
a) Demonstrate proficiency in the foundation areas of Computer Science including mathematics, discrete structures, logic and the theory of algorithms	
b) Demonstrate proficiency in various areas of Computer Science including data structures and algorithms, concepts of programming languages and computer systems.	3, 4
c) Demonstrate proficiency in problem solving and application of software engineering techniques	1, 2, 3, 4, 5, 6, 7
d) Demonstrate mastery of at least one modern programming language and proficiency in at least one other.	
e) Demonstrate understanding of the social and ethical concerns of the practicing computer scientist.	
f) Demonstrate the ability to work cooperatively in teams.	2, 5, 7
g) Demonstrate effective communication skills.	2, 5, 6, 7, 8

Assessment Plan for the Course & how Data in the Course are used to assess Program Outcomes

Student and Instructor Course Outcome Surveys are administered at the conclusion of each offering, and are evaluated as described in the School's Assessment Plan: http://www.cis.fiu.edu/programs/undergrad/cs/assessment/

Outline

Topic	Number of Lecture Hours	Course Outcome
SE/Software Processes • Systems level considerations, i.e., the interaction of software with its intended environment (cross- reference IAS/Secure Software Engineering) • Introduction to software process models (e.g., waterfall, incremental, agile) o Activities within software lifecycles • Programming in the large vs. individual programming • Evaluation of software process models SE/Tools and Environment • Software configuration management and version control • Requirements analysis and design modeling tools SE/Software Project Management • Team participation ○ Team processes including responsibilities for tasks, meeting structure, and work schedule ○ Roles and responsibilities in a software team ○ Team conflict resolution ○ Risks associated with virtual teams (communication, perception, structure) • Team management ○ Team organization and decision-making ○ Role identification and assignment ○ Individual and team performance assessment • Project management ○ Scheduling and tracking ○ Project management tools	7	1, 5, 6, 7
SE/Software Design • Structural and behavioral models of software designs	8	1, 2, 6
SE/Requirements Engineering • Describing functional requirements using, for example, use cases or users stories • Properties of requirements including consistency, validity, completeness, and feasibility • Software requirements elicitation • Describing system data using, for example, class diagrams or entity-relationship diagrams • Non-functional requirements and their relationship to software quality (cross-reference IAS/Secure Software Engineering)	12	1, 2, 5, 6, 7
SE/Software Design • System design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures • Design Paradigms such as structured design (top-down functional decomposition), object-oriented analysis and design, event driven design, component-level design, data-structured centered, aspect oriented, function oriented, service oriented • Design patterns • Relationships between requirements and designs: transformation of models,	10	1, 3, 5, 6, 7

design of contracts, invariants		
SE/Software Validation • Verification and validation concepts • Inspections, reviews, audits • Testing fundamentals (cross-reference SDF/Development Methods) ○ Unit, integration, validation, and system testing ○ Test plan creation and test case generation ○ Black-box and white-box testing techniques ○ Regression testing and test automation	4	4, 5

*Topics derived from: The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.

Learning Outcomes: (Familiarity→Usage→Assessment)

SE/Software Processes:

1. Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems. [Familiarity]
2. Describe the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile). [Familiarity]
3. Describe the different practices that are key components of various process models. [Familiarity]
4. Differentiate among the phases of software development. [Familiarity]
5. Describe how programming in the large differs from individual efforts with respect to understanding a large
   code base, code reading, understanding builds, and understanding context of changes. [Familiarity]
6. Explain the concept of a software lifecycle and provide an example, illustrating its phases including the deliverables that are produced. [Familiarity]

Requirements Engineering:

1. List the key components of a use case or similar description of some behavior that is required for a system. [Familiarity]
2. Describe how the requirements engineering process supports the elicitation and validation of behavioral requirements. [Familiarity]
3. Interpret a given requirements model for a simple software system. [Familiarity]
4. Identify both functional and non-functional requirements in a given requirements specification for a software system. [Usage]

Software Design:

1. Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation. [Familiarity]
2. Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design. [Usage]
3. Construct models of the design of a simple software system that are appropriate for the paradigm used to design it. [Usage]
4. Within the context of a single design paradigm, describe one or more design patterns that could be applicable to the design of a simple software system. [Familiarity]
5. Create appropriate models for the structure and behavior of software products from their requirements specifications. [Usage]
6. Explain the relationships between the requirements for a software product and its design, using appropriate models. [Assessment]

SE/Software Verification and Validation

1. Distinguish between program validation and verification. [Familiarity]
2. Undertake, as part of a team activity, an inspection of a medium-size code segment. [Usage]
3. Describe and distinguish among the different types and levels of testing (unit, integration, systems, and acceptance). [Familiarity]
4. Create and document a set of tests for a medium-size code segment. [Usage]

Knight Foundation School of Computing and Information Sciences
CEN 4010
Software Engineering I

SE/Tools and Environment

1. Describe how version control can be used to help manage software release management. [Familiarity]

SE/Software Project Management

1. Discuss common behaviors that contribute to the effective functioning of a team. [Familiarity]
2. Create and follow an agenda for a team meeting. [Usage]
3. Identify and justify necessary roles in a software development team. [Usage]
4. Understand the sources, hazards, and potential benefits of team conflict. [Usage]
5. Apply a conflict resolution strategy in a team setting. [Usage]
6. Use an *ad hoc* method to estimate software development effort (e.g., time) and compare to actual effort required. [Usage]
7. Demonstrate through involvement in a team project the central elements of team building and team management. [Usage]
8. Create a team by identifying appropriate roles and assigning roles to team members. [Usage]
9. Assess and provide feedback to teams and individuals on their performance in a team setting. [Usage]
10. Track the progress of some stage in a project using appropriate project metrics. [Usage]

*Learning Outcomes drawn from : The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.

Course Outcomes Emphasized in Laboratory Projects / Assignments

Course Outcome	Number of Weeks
1. Software Requirement and Analysis Model Outcomes: 1,2,6,7	4
2. Software Design Document Outcomes: 1,3,6,7	4
3. Final Software Project Demonstration Outcomes: 1,2,3,4,5,6,7	4

Oral and Written Communication:

Written Reports		Oral Presentations	
Number Required	Approx. Number of pages for each	Number Required	Approx. Time for each
3 (Software Requirement, Design Document and Final System Document)	30	2	15 minutes per group (5 minutes per student)

Knight Foundation School of Computing and Information Sciences
CEN 4010
Software Engineering I

Social and Ethical Implications of Computing Topics:

No significant coverage

Topic	Class time	Student Performance Measures

Approximate number of credit hours devoted to fundamental CS topics

Topic	Core Hours	Advanced Hours
Algorithms:		
Software Design:		2.0
Computer Organization and Architecture:		
Data Structures:		
Concepts of Programming Languages:		

Theoretical Contents

Topic	Class time
Invariants, pre and post conditions	1.0

Problem Analysis Experiences

Software requirement and analysis model

Solution Design Experiences

1. System Design using Architectural Patterns
2. Detailed Object design using Design Patterns

Knight Foundation School of Computing and Information Sciences
CEN 4010
Software Engineering I

The Coverage of Knowledge Units within Computer Science Body of Knowledge[1]

KA	KUnit	Learning Outcome	Topic	Type	Lecture Hours
SE	Software Processes	1, 2,3,4,5,6	• Systems level considerations, i.e., the interaction of software with its intended environment (cross- reference IAS/Secure Software Engineering) • Introduction to software process models (e.g., waterfall, incremental, agile) ○ Activities within software lifecycles • Programming in the large vs. individual programming	[Core-Tier1]	2
		6	• Evaluation of software process models	[Core-Tier2]	1
SE	Software Requirements Engineering	1,2,3	• Describing functional requirements using, for example, use cases or users stories • Properties of requirements including consistency, validity, completeness, and feasibility	[Core-Tier1]	6
		6	• Software requirements elicitation • Describing system data using, for example, class diagrams or entity-relationship diagrams • Non-functional requirements and their relationship to software quality (cross-reference IAS/Secure Software Engineering)	[Core-Tier2]	6
SE	Software Design	1,2,3,4	• System design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures • Design Paradigms such as structured design (top-down	[Core-Tier1]	10

[1]*Knowledge Unit drawn from : The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

			functional decomposition), object-oriented analysis and design, event driven design, component-level design, data-structured centered, aspect oriented, function oriented, service oriented • Structural and behavioral models of software designs • Design patterns		
		6,7	• Relationships between requirements and designs: transformation of models, design of contracts, invariants	[Core-Tier2]	8
SE	Software Verification and Validation	3,4,6	• Verification and validation concepts • Inspections, reviews, audits • Testing fundamentals (cross-reference SDF/Development Methods) ○ Unit, integration, validation, and system testing ○ Test plan creation and test case generation ○ Black-box and white-box testing techniques ○ Regression testing and test automation	[Core-Tier2]	4
SE	Tools and Environment	2	• Software configuration management and version control • Requirements analysis and design modeling tools	[Core-Tier2]	1
SE	Software Project Management	1,2,3,4,5,6	• Team participation ○ Team processes including responsibilities for tasks, meeting structure, and work schedule ○ Roles and responsibilities in a software team ○ Team conflict resolution ○ Risks associated with virtual teams (communication, perception, structure)	[Core-Tier2]	2
		10,12,13,15	• Team management • Team organization and decision-making • Role identification and assignment • Individual and team performance assessment • Project management • Scheduling and tracking o Project management tools	[Elective]	1