CS2013

Software Engineering (SE)

Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building reliable software systems that satisfy the requirements of users and customers. It is applicable to small, medium, and large-scale systems. It and encompasses all phases of the life cycle of a software system. The life cycle includes requirements elicitation, analysis and specification; design; construction; verification and validation; deployment; and operation and maintenance. 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 (or teams) of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment. The software engineering toolbox has evolved over the years; for instance, the use of contracts (such as a 'requires' clause, an 'ensures' clause, and class invariants) is now regarded as good practice.

The elements of software engineering are applicable to the development of software in any computing application domain where professionalism, quality, schedule, and cost are important in producing a software system. In the past decade, a wide variety of software engineering practices have been developed and utilized with a number of tradeoffs identified. It is now necessary for a practicing software engineering to apply appropriate techniques and practices to a given development effort in order to maximize value.

Students and instructors need to appreciate the impacts of the specialization on software engineering approaches. Specialized systems include:

- Real-time systems
- Client-server systems
- Distributed systems
- Parallel systems
- Web-based systems
- High-integrity systems
- Games
- Mobile computing
- Domain-specific software (e.g., scientific computing)

The issues raised in each of these specializations demand specific treatments in each of the phases of software engineering. Students should be made aware of the differences between general software engineering techniques and principles and techniques and principles developed to address specialization-specific issues.

In general, students best learn much of the material defined in the SE KA at the application level by participating in a project that requires them to work on a team to develop a software system through as much of its lifecycle as is possible. Much of software engineering is devoted to effective communication among team members and stakeholders. Utilizing project teams makes it possible projects to be sufficiently challenging to require use of effective software engineering techniques and also requires that students develop and practice their communication skills.

SE. Software Engineering (5 Core-Tier1 hours; 21 Core-Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
SE/SoftwareProcesses	**1**	**2**	**Y**
SE/SoftwareProjectManagement		**3**	**Y**
SE/ToolsAndEnvironments		**2**	**N**
SE/RequirementsEngineering		**3**	**Y**
SE/SoftwareDesign	**4**	**4**	**Y**
SE/SoftwareConstruction		**2**	**Y**
SE/SoftwareVerificationValidation		**3**	**Y**
SE/SoftwareEvolution		**1**	**Y**
SE/FormalMethods			**Y**
SE/SoftwareReliability		**1**	**Y**

SE/SoftwareProcesses [1 Core-Tier1 hours; 2 Core-Tier2 hours]

Topics:

Core Tier 1:

- Systems level considerations, i.e., the interaction of software with its intended environment
- Phases of software life-cycles
- Programming in the large vs. individual programming

Core Tier 2:

- Software process models (e.g., waterfall, incremental, agile)

Elective

- Process improvement
- Software process capability maturity models
- Software process measurements

Learning Outcomes:

Tier 1 Outcomes:

1. Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems. [Knowledge]
2. Differentiate among the phases of software development. [Knowledge]
3. Explain the concept of a software life cycle and provide an example, illustrating its phases including the deliverables that are produced. [Knowledge]
4. 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. [Knowledge]

Tier 2 Outcomes:

1. Compare several common process models with respect to their value for development of particular classes of software systems taking into account issues such as requirement stability, size, and non-functional characteristics. [Application]

Elective Outcomes:

1. Describe the intent and fundamental similarities among process improvement approaches. [Knowledge]
2. Compare several process improvement models such as CMM, CMMI, CQI, Plan-Do-Check-Act, or ISO9000. [Knowledge]
3. Use a process improvement model such as PSP to assess a development effort and recommend approaches to improvement. [Application]
4. Explain the role of process maturity models in process improvement. [Knowledge]
5. Describe several process metrics for assessing and controlling a project. [Knowledge]
6. Use project metrics to describe the current state of a project. [Application]

SE/SoftwareProjectManagement [3 Core-Tier2 hours]

Topics:

Core Tier 2:

- Risk
  - The role of risk in the life cycle
  - Risk categories including security, safety, market, financial, technology, people, quality, structure and process
  - Risk identification
  - Risk tolerance (e.g., risk-adverse, risk-neutral, risk-seeking)
  - Risk removal, reduction and control
- 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)
- Effort Estimation (at the personal level)

Elective:

1. Team management
   - Team organization and decision-making
   - Role identification and assignment
   - Individual and team performance assessment
2. Project scheduling and tracking
3. Software measurement and estimation techniques
4. Software quality assurance and the role of measurements
5. Project management tools
6. Principles of risk management
7. Risk analysis and evaluation
8. Cost/benefit analysis
9. System-wide approach to risk including hazards associated with tools

Learning Outcomes:

Core (Tier 2) Outcomes

1. List several examples of software risks. [Knowledge]
2. Describe the impact of risk in a software development life cycle. [Knowledge]
3. Describe different categories of risk in software systems. [Knowledge]
4. Describe the impact of risk tolerance on the software development process. [Application]
5. Identify risks and describe approaches to managing risk (avoidance, acceptance, transference,

mitigation), and characterize the strengths and shortcomings of each. [Knowledge]
6. Explain how risk affects decisions in the software development process. [Application]
7. Identify behaviors that contribute to the effective functioning of a team. [Knowledge]
8. Create and follow an agenda for a team meeting. [Application]
9. Identify and justify necessary roles in a software development team. [Application]
10. Understand the sources, hazards, and potential benefits of team conflict. [Application]
11. Apply a conflict resolution strategy in a team setting. [Application]
12. Use an *ad hoc* method to estimate software development effort (e.g., time) and compare to actual effort required. [Application]

Elective Outcomes:
1. Identify security risks for a software system. [Application]
2. Demonstrate through involvement in a team project the central elements of team building and team management. [Application]
3. Identify several possible team organizational structures and team decision-making processes. [Knowledge]
4. Create a team by identifying appropriate roles and assigning roles to team members. [Application]
5. Assess and provide feedback to teams and individuals on their performance in a team setting. [Application]
6. 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. [Application]
7. Track the progress of a project using appropriate project metrics. [Application]
8. Compare simple software size and cost estimation techniques. [Application]
9. Use a project management tool to assist in the assignment and tracking of tasks in a software development project. [Application]
10. Demonstrate a systematic approach to the task of identifying hazards and risks in a particular situation. [Application]
11. Apply the basic principles of risk management in a variety of simple scenarios including a security situation. [Application]
12. Conduct a cost/benefit anaysis for a risk mitigation approach. [Application]
13. Identify and analyse some of the risks for an entire system that arise from aspects other than the software.

SE/ToolsAndEnvironments [2 Core-Tier2 hours]

Topics:
Core Tier 2:
- Software configuration management and version control; release management
- Requirements analysis and design modeling tools
- Testing tools including static and dynamic analysis tools
- Programming environments that automate parts of program construction processes (e.g., automated builds)
- Tool integration mechanisms

Learning Outcomes:
Core Tier 2:
1. Identify configuration items and use a source code control tool in a small team-based project. [Application]
2. Select, with justification, an appropriate set of tools to support the development of a range of software products including tools for requirements tracking, design modeling, implementation, build automation, and testing. [Application]
3. Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size. [Application]

SE/RequirementsEngineering [3 Core-Tier2 hours]

Topics:
Core Tier 2:
- Properties of requirements including consistency, validity, completeness, and feasibility
- Software requirements elicitation
- Functional and non-functional requirements
- Describing functional requirements using, for example, use cases or users stories
- Describing system data using, for example, class diagrams or entity-relationship diagrams
- Evaluation and use of requirements specifications

Elective:
- Requirements analysis modeling techniques
- Acceptability of certainty / uncertainty considerations regarding software / system behaviour
- Prototyping
- Basic concepts of formal requirements specification
- Requirements specification
- Requirements validation
- Requirements tracing

Learning Outcomes:
Core (Tier 2) Outcomes:
1. Conduct a review of a set of software requirements to determine the quality of the requirements with respect to the characteristics of good requirements. [Application]
2. Describe the fundamental challenges of and common techniques used for requirements elicitation. [Knowledge]
3. List the key components of a use case or similar description of some behavior that is required for a system and discuss their role in the requirements engineering process. [Knowledge] List the key components of a class diagram or similar description of the data that a system is required to handle. [Knowledge]
4. Identify both functional and non-functional requirements in a given requirements specification for a software system. [Application]
5. Use a software requirement specification as the basis of a software development effort. [Application]

Elective Outcomes:
1. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system. [Application]
2. 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 [Application]
3. Translate into natural language a software requirements specification (e.g., a software component contract) written in a formal specification language. [Application]
4. Create a prototype of a software system to mitigate risk in requirements. [Application]
5. Differentiate between forward and backward tracing and explain their roles in the requirements validation process. [Knowledge]

SE/SoftwareDesign [4 Core-Tier1 hours; 4 Core-Tier2 hours]
Topics:
Core Tier 1
- Overview of design paradigms
- System design principles: divide and conquer (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures.
- Appropriate models of software designs, including structure and behavior.
- Software architecture concepts

Core Tier 2

- 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.
- Relationships between requirements and designs: transformation of models, design of contracts.
- Architectural design: standard architectures (eg client-server, n-layer, transform center, pipes-and-filters, etc), analysis and design patterns, refactoring of designs.
- The use of components in design: component selection, design, adaptation and assembly of components, components and patterns, components and objects, (for example, build a GUI using a standard widget set).

Elective

- Internal design qualities, and models for them: efficiency and performance, redundancy and fault tolerance, traceability of requirements.
- External design qualities, and models for them: functionality, reliability, performance and efficiency, usability, maintainability, portability.
- Measurement and analysis of design quality.
- Tradeoffs between different aspects of quality.
- Application frameworks.
- Middleware: the object-oriented paradigm within middleware, object request brokers and marshalling, transaction processing monitors, workflow systems.

Learning Outcomes:

Core (Tier 1) Outcomes:

1. Articulate design principles including separation of concerns, information hiding, coupling and cohesion, and encapsulation.[Knowledge]
2. Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design. [Application]
3. Construct models of the design of a simple software system that are appropriate for the paradigm used to design it. [Application]
4. For the design of a simple software system within the context of a single design paradigm, describe the software architecture of that system. [Knowledge]
5. 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. [Knowledge]
6. Given a high-level design, identify the software architecture by differentiating among common software architectures such as 3-tier, pipe-and-filter, and client-server. [Knowledge]

Core (Tier 2) Outcomes:

1. For a simple system suitable for a given scenario, discuss and select an appropriate design paradigm. [Application]
2. Create appropriate models for the structure and behavior of software products from their requirements specifications. [Application]
3. Explain the relationships between the requirements for a software product and the designed structure and behavior, in terms of the appropriate models and transformations of them. [Evaluation]
4. Apply simple examples of patterns in a software design. [Application]
5. Investigate the impact of software architectures selection on the design of a simple system.
6. Select suitable components for use in the design of a software product. [Application]
7. Explain how suitable components might need to be adapted for use in the design of a software product. [Knowledge].
8. Design a contract for a typical small software component for use in a given system. [Application]

Elective Outcomes:

1. Discuss and select an appropriate software architecture for a simple system suitable for a given

scenario. [Application]
2. Apply models for internal and external qualities in designing software components to achieve an acceptable tradeoff between conflicting quality aspects. [Application]
3. Analyse a software design from the perspective of a significant internal quality attribute. [Evaluation]
4. Analyse a software design from the perspective of a significant external quality attribute. [Evaluation]
5. Explain the role of objects in middleware systems and the relationship with components. [Knowledge]
6. Apply component-oriented approaches to the design of a range of software, such as using components for concurrency and transactions, for reliable communication services, for database interaction including services for remote query and database management, or for secure communication and access. [Application]

SE/SoftwareConstruction [2 Core-Tier2 hours]

Topics:

Core Tier 2:

- Coding practices: techniques, idioms/patterns, mechanisms for building quality programs
  - Defensive coding practices
  - Secure coding practices
  - Using exception handling mechanisms to make programs more robust, fault-tolerant
- Coding standards
- Integration strategies

Elective:

RobustAndSecurityEnhancedProgramming

- Defensive programming
  - Principles of secure design and coding:
  - Principle of least privilege
  - Principle of fail-safe defaults
  - Principle of psychological acceptability
- Potential security problems in programs
  - Buffer and other types of overflows
  - Race conditions
  - Improper initialization, including choice of privileges
  - Checking input
  - Assuming success and correctness
  - Validating assumptions
- Documenting security considerations in using a program

Learning Outcomes:

Core (Tier 2) Outcomes:

1. Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness. [Knowledge]
2. Build robust code using exception handling mechanisms. [Application]
3. Describe secure coding and defensive coding practices. [Knowledge]
4. Select and use a defined coding standard in a small software project. [Application]
5. Compare and contrast integration strategies including top-down, bottom-up, and sandwich integration. [Knowledge]

Elective Outcomes:

1. Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions
2. State and apply the principles of least privilege and fail-safe defaults.
3. Write a simple library that performs some non-trivial task and will not terminate the calling

program regardless of how it is called

SE/SoftwareVerificationValidation [3 Core-Tier2 hours]

Topics:

Core Tier 2:

- Verification and validation
- Testing types, including human computer interface, usability, reliability, security, conformance to specification
- Testing fundamentals
  - Unit, integration, validation, and system testing
  - Test plan creation and test case generation
  - Black-box and white-box testing techniques
- Defect tracking
- Testing parallel and distributed systems

Elective*:*

- Static approaches and dynamic approaches to verification
- Regression testing
- Validation planning; documentation for validation
- Object-oriented testing; systems testing
- Verification and validation of non-code artifacts (documentation, help files, training materials)
- Fault logging, fault tracking and technical support for such activities
- Fault estimation and testing termination including defect seeding
- Inspections, reviews, audits

Learning Outcomes:

Core Tier 2 Outcomes:

1. Distinguish between program validation and verification. [Knowledge]
2. Describe the role that tools can play in the validation of software. [Knowledge]
3. Describe the different types of testing. [Knowledge]
4. Distinguish between the different types and levels of testing (unit, integration, systems, and acceptance). [Knowledge]
5. Describe techniques for identifying significant test cases for unit, integration, and system testing. [Knowledge]
6. Use a defect tracking tool to manage software defects in a small software project. [Application]
7. Describe the issues and approaches to testing distributed and parallel systems. [Knowledge]

Elective Outcomes:

1. Create, evaluate, and implement a test plan for a medium-size code segment. [Application]
2. Undertake, as part of a team activity, an inspection of a medium-size code segment. [Application]
3. Compare static and dynamic approaches to verification. [Knowledge]
4. Discuss the issues involving the testing of object-oriented software. [Application]
5. Describe techniques for the verification and validation of non-code artifacts. [Knowledge]
6. Describe approaches for fault estimation. [Knowledge]
7. Estimate the number of faults in a small software application based on fault density and fault seeding. [Application]
8. Conduct an inspection or review of software source code for a small or medium sized software project. [Application]

SE/SoftwareEvolution [1 Core-Tier2 hour]

Topics:

- Software development in the context of large, pre-existing code bases
- Software evolution
- Characteristics of maintainable software

- Reengineering systems
- Software reuse

Learning Outcomes:
Core Tier 2:
1. Identify the principal issues associated with software evolution and explain their impact on the software life cycle. [Knowledge]
2. Discuss the challenges of evolving systems in a changing environment. [Knowledge]
3. Outline the process of regression testing and its role in release management. [Application]
4. Discuss the advantages and disadvantages of software reuse. [Knowledge]

Elective Outcomes:
1. Estimate the impact of a change request to an existing product of medium size. [Application]
2. Identify weaknesses in a given simple design, and removed them through refactoring. [Application]

SE/FormalMethods [elective]

The topics listed below have a strong dependency on core material from the Discrete Structures area, particularly knowledge units DS/FunctionsRelationsAndSets, DS/BasicLogic and DS/ProofTechniques.
***Topics*:**
· Role of formal specfication and analysis techniques in the software development cycle
· Program assertion languages and analysis approaches (including languages for writing and analyzing pre and postconditions, such as OCL, JML)
· Formal approaches to software modeling and analysis
- Model checkers
- Model finders

· Tools in support of formal methods

Learning Outcomes:
1. Describe the role formal specification and analysis techniques can play in the development of complex software and compare their use as validation and verification techniques with testing., [Knowledge]
2. Apply formal specification and analysis techniques to software designs and programs with low complexity. [Application]
3. Explain the potential benefits and drawbacks of using formal specification languages. [Knowledge]
4. Create and evaluate program assertions for a variety of behaviors ranging from simple through complex. [Application]
5. Using a common formal specification language, formulate the specification of a simple software system and derive examples of test cases from the specification. [Application]

SE/SoftwareReliability [1 core tier-2, Elective]

Topics:
Core Tier-2
- Software reliability engineering concepts
- Software reliability, system reliability and failure behavior
- Fault lifecycle concepts and techniques

Elective
- Software reliability models
- Software fault tolerance techniques and models
- Software reliability engineering practices
- Measurement-based analysis of software reliability

Learning Outcomes:
Core Tier 2:

1. Explain the problems that exist in achieving very high levels of reliability. [Knowledge]
2. Describe how software reliability contributes to system reliability [Knowledge]
3. List approaches to minimizing faults that can be applied at each stage of the software lifecycle. [Knowledge]

Elective Outcomes:

1. Compare the characteristics of three different reliability modeling approaches. [Knowledge]
2. Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system. [Application]
3. Identify methods that will lead to the realization of a software architecture that achieves a specified reliability level of reliability. [Application]
4. Identify ways to apply redundancy to achieve fault tolerance for a medium-sized application. [Application]