Software Engineering (CE - SWE)

CE-SWE0. History and overview of software engineering
CE-SWE1. Languages for the computer engineer
CE-SWE2. Software processes [core]
CE-SWE3. Software requirements and specifications [core]
CE-SWE4. Software design [core]
CE-SWE5. Software testing and validation [core]
CE-SWE6. Software evolution [core]
CE-SWE7. Software tools and environments [core]
CE-SWE8. Language translation [elective]
CE-SWE9. Software project management [elective]

Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building 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 creation of programs benefits from concepts and practices from software engineering. There is scope for introducing fundamental ideas from software engineering into elementary programming and into early experience of software design.

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.

CE-SWE0. History and overview of software engineering [core]

Suggested time: 1 hour

Topics:
- 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.

Learning objectives:
Identify some contributors to software engineering and relate their achievements to the knowledge area.
Provide examples of the software process.
Articulate the difference between software engineering and computer engineering.
Articulate some of the components of a software process.
Provide some examples that would use software engineering.
Give reasons for the importance of testing and validation in the development of software.
Describe how computer engineering uses or benefits from software engineering.

CE-SWE 1. Languages for the computer engineer [core]
Minimum core coverage time: 2 hours

Topics:
- History of languages and their role in computer engineering
- The range and nature of the languages employed within the framework of computer engineering: to include programming, specification, scripting, web based, design, hardware description languages.
- The common features of languages: abstraction, typing where relevant, components. Modules, procedures, functions as abstraction mechanisms. Parameterisation mechanisms – for modules, procedures, functions
- Activation records and storage management
- Comparisons, including similarities. Large scale, small scale issues.
- The role of software support and software tools.
- The effects of scale on methodologies

Learning outcomes:
1. Summarize the evolution of languages in computer engineering illustrating how this history has led to the paradigms available today.
2. Explain how abstraction mechanisms support the creation of reusable software components.
3. Compare and contrast the role that tools play in the different circumstances.
4. Distinguish between features that support in-the-small-activity and in-the-large-activity.

**CE-SWE2. Software processes [core]**

*Suggested time: 2 hours*

*Topics:*
- Software life cycle and process models
- Process assessment models
- Software process metrics

*Learning objectives:*
1. Select, with justification, the software development models most appropriate for the development and maintenance of diverse software products.
2. Explain the role of process maturity models.

**CE-SWE3. Software requirements and specifications [core]**

*Suggested time: 6 hours*

*Topics:*
- Requirements elicitation
- Requirements analysis modeling techniques
- Functional and nonfunctional requirements
- Prototyping
- Basic concepts of formal specification techniques

*Learning objectives:*
1. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system.
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 (e.g., structured analysis or object-oriented-analysis).
3. Conduct a review of a software requirements document using best practices to determine the quality of the document.
4. Translate into natural language a software requirements specification written in a commonly used formal specification language.

**CE-SWE4. Software design [core]**

*Suggested time: 6 hours*

*Topics:*
- Fundamental design concepts and principles
- Software architecture
Structured design  
Object-oriented analysis and design  
Component-level design  
Design for reuse

Learning objectives:
1. Evaluate the quality of multiple software designs based on key design principles and concepts.
2. 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).
3. Using appropriate guidelines, conduct the review of a software design.

CE-SWE5.  Software testing and validation [core]

Suggested time: 6 hours

Topics:
- 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

Learning objectives:
1. Demonstrate the application of the different types and levels of testing (unit, integration, systems, and acceptance) to software products of medium size.
2. Undertake, as part of a team activity, an inspection of a medium-size code segment.
3. Describe the role that tools can play in the validation of software.

CE-SWE6   Software evolution [core]

Suggested time: 4 hours

Topics:
- Software maintenance: the different forms of maintenance; the associated disciplines 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
Learning objectives:
1. Identify the principal issues associated with software evolution and explain their impact on the software life cycle.
2. Develop a plan for re-engineering a medium-sized product in response to a change request.
3. Discuss the advantages and disadvantages of software reuse.
4. Demonstrate the ability to exploit opportunities for software reuse in a variety of contexts.

CE-SWE7. Software tools and environments [core]

Suggested time: 2 hours

Topics:
- 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

Learning objectives:
1. Select, with justification, an appropriate set of tools to support the software development of a range of software products.
2. Analyze and evaluate a set of tools in a given area of software development (e.g., management, modeling, or testing).
3. Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.

CE-SWE8. Language translation [elective]

Suggested time: 5 hours

Topics:
- 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, optimisation); separate compilation or translation - the benefits and the mechanisms; machine-dependent and machine-independent aspects of translation

Learning outcomes:
1. Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
2. Describe the phases of program translation from source code to executable code and the files produced by these phases.
3. Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.

CE-SWE9. Software project management [elective]

Suggested time: 4 hours

Topics:
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

Learning objectives:
1. Demonstrate through involvement in a team project the central elements of team building and team management.
2. 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.
3. Compare and contrast the different methods and techniques used to assure the quality of a software product.