Introduction to Software Engineering Computing

Description

This course covers the basic principles and concepts of software engineering; system requirements; secure programming in the large; modeling and testing; object oriented analysis and design using the UML; design patterns; frameworks and APIs; client-server architecture; user interface technology; and the analysis, design and programming of extensible software systems.

Sample labs and assignments:
- Evaluating the performance of various simple software designs.
- Adding features to an existing system.
- Testing a system to verify conformance to test cases.
- Building a GUI for an application.
- Numerous exercises building models in UML, particularly class diagrams and state machines.
- Developing and presenting a simple set of requirements (to be done as a team) for some innovative client server application of very small size.
- Implementing the above, using reusable technology to the greatest extent possible.

Additional teaching considerations:
- This course is a good starting point for exposing students to moderately sized existing systems. With such systems, they can learn and practice the essential skills of reading and understanding code written by others. Students should write code in the context of a particular domain, for example the biological, physical, mathematical or chemical sciences or even wider spectra such as game programming, business applications, and graphics and animation.
- It is suggested that a core subset of UML be taught, rather than trying to cover all features.
- It may be challenging for instructors to convey the nature of SE to students; however, this challenge may be addressed through strategies such as field trips to businesses and industries that utilize large software systems, guest lectures by developers and users of large software systems, and discussions about embedded systems in everyday life including ATMs, wireless devices, cell phones, various mobile devices, and computer games.

Minimum Contact Hours

42

Prerequisite(s)

Computer Science II

Corequisite(s)

Computer Science III
Topics

<table>
<thead>
<tr>
<th>Title</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design patterns</td>
<td>8</td>
<td>abstraction-occurrence, composite, player-role, singleton, observer, delegation, facade, adapter, etc.</td>
</tr>
<tr>
<td>Domain modeling</td>
<td>3</td>
<td>examples of building class diagrams to model various domains</td>
</tr>
<tr>
<td>Introduction to software architecture</td>
<td>3</td>
<td>client-server computing, architectural patterns, such as pipe &amp; filter and MVC</td>
</tr>
<tr>
<td>Introduction to software project management</td>
<td>5</td>
<td>project planning, estimation, configuration management, risk management, and software security process models, such as Building Security In Maturity Model (BSIMM), OWASP Software Assurance Maturity Model (SAMM), and Microsoft Software Development Lifecycle (SDL)</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>3</td>
<td>requirements analysis</td>
</tr>
<tr>
<td>Reusable technologies</td>
<td>3</td>
<td>reusable technologies as a basis for software engineering, frameworks and APIs, risks associated with reuse, such as the Ariane rocket experience</td>
</tr>
<tr>
<td>Role of assured software engineering</td>
<td>1</td>
<td>software engineering for assurance and its place as an engineering discipline</td>
</tr>
<tr>
<td>Software behavior</td>
<td>7</td>
<td>correctness under all conditions of use and representing software behavior using sequence diagrams, state machines and activity diagrams</td>
</tr>
<tr>
<td>Unified Modeling Language (UML)</td>
<td>3</td>
<td>review of object-oriented principles, UML class diagrams, and object-oriented analysis</td>
</tr>
<tr>
<td>Use and misuse cases</td>
<td>3</td>
<td>use cases, misuse cases, and user-centered design</td>
</tr>
<tr>
<td>Verification and validation</td>
<td>3</td>
<td>inspections and reviews, integration, system and acceptance testing.</td>
</tr>
</tbody>
</table>

Course Objectives

Global Impact

An ability to analyze the global impact of computing on individuals, organizations, and society.

- **Learning Outcomes**
  CS. 3.
  Practice the tenets of ethics and professional behavior promoted by computing societies; accept the professional responsibilities and liabilities associated with software development.

Lifelong Learning

An ability to engage in continuous learning as well as research and assess new ideas and information to provide the capabilities for lifelong learning.

- **Learning Outcomes**
  SE. 2.
  Construct a preliminary investigative report for a proposed system that includes scheduling and plans for mitigating potential risks.
Create effective, efficient and secure software, reflecting standard principles of software engineering and software assurance.

Decompose complex systems using best practice object-oriented analysis and design tools and techniques.

Design and implement consistent and coherent user-centered interfaces that comply with UI standard practices.

Evaluate and test software system requirements that considers both validation and verification.

Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software application that is fully implemented and formally tested.