Computer Science II Computing

Description

This course is the first in a three-course sequence that provides students with a foundation in computer science. Students develop fundamental programming skills using a language that supports an object-oriented approach, incorporating security awareness, human-computer interactions and social responsibility.

Among the associate-degree transfer programs, the Computer Science I-II-III coursework provides a common "core" body of knowledge in computing. The complete course sequence is designed in such a manner that students progress in knowledge, proficiency and professional maturity in several specific areas, including software engineering principles and professional and ethical conduct.

The progression of software engineering topics across CS I-II-III originates in CS I, where there is an emphasis on using a cyclic approach for program development by iterating through designing, coding, and testing program modules. Complemented by algorithm analysis, students are encouraged to think abstractly about problems and to begin developing processes for decomposing problems into organized parts. Encouraging clear documentation, good naming conventions and consistent secure coding style contribute to a disciplined approach to writing programs.

The progression of software engineering topics across CS I-II-III continues in CS II, where greater emphasis is placed on abstraction and sound software design principles, engaging students in the development of secure software components that solve a wide range of related problems and can be reused. The students determine the necessary elements of simple ADTs (such as a counter or a date) and then construct them; by their very nature, these components must be well-documented to encourage reuse. Additionally the students write assertions such as pre-conditions and post-conditions describing each class method, thereby encouraging students to think deeply about a simple problem before coding. After coding, the components must be well-tested, and therefore the use of test plans and test drivers are practiced. These activities reinforce the notion of constructing software from well-defined, independent pieces and complement the study of using existing library classes and APIs in software solutions.

The progression of software engineering topics across CS I-II-III concludes in CS III, where students are asked to step beyond the programmer role and take a broader view of software development; to consider its lifecycle from problem description to maintenance. Students first practice with analysis and design of medium-sized systems. Standard modeling tools are introduced and the students complete the phases of analysis, design, implementation and testing of a medium-sized team project that includes documents such as UML diagrams or CRC cards in addition to test plans. Students consider design patterns and write applications using data structures and templates. The software engineering topics are integrated with professionalism and ethics, as well as software and information assurance topics, such as security concerns and liabilities of computer-based systems.
The progression of the emphasis on professional and ethical conduct across CS I-II-III originates in CS I, where the curriculum is designed to consider the historical context of computing and programming as well as examining issues involving ethical conduct; plagiarism, intellectual property and software piracy issues are presented. Typically, student requirements for submitting original work as well as college policies regarding the use of computing resources and acceptable computing behavior on campus and the Brookings Institute "10 Commandments of Ethical Computing" can be used as relevant discussion starters.

The progression of the emphasis on professional and ethical conduct across CS I-II-III continues in CS II, which builds upon this foundation by examining societal issues, the Internet, and professionalism. Now that the students have gained some experience with developing programs, they can begin to see "what can go wrong" and the possible consequences to the user of their program, at a personal level, such as infinite loops and program crashes. Additionally, students are confronted with broader implications by design considerations regarding databases and data accessibility; ethical concerns regarding personal data, privacy and property rights should be explored. Integrating these topics with the software development process, security issues, and relevant cases of software errors will help students recognize that their work can have individual as well as societal consequences and encourage them to think carefully about the design and implementation of their programs.

The progression of the emphasis on professional and ethical conduct across CS I-II-III concludes in CS III, where a broader view is presented - encompassing computing sciences as a profession. Standards of professional behavior, organizations and publications are examined as well as a variety of occupational roles in the computing field. Course content materials presenting case studies of significant software failures amplify the topics of risks and liabilities. The students should start to recognize that invariably software production involves ethical choices. Incorporating these topics with the software lifecycle, engineering, human factors, and software assurance considerations will help students internalize the significance of professional and ethical behavior and subsequently demonstrate it through their individual and group projects.

**Minimum Contact Hours**

42

**Prerequisite(s)**

Computer Science I

**Corequisite(s)**

Discrete Structures

**Topics**

<table>
<thead>
<tr>
<th>Title</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event-driven programming</td>
<td>4</td>
<td>graphics API; event creation; event-handling methods; exception handling</td>
</tr>
<tr>
<td>Intermediate computing algorithms</td>
<td>5</td>
<td>searching; sorting; recursive algorithms; complexity of algorithms</td>
</tr>
<tr>
<td>Intermediate data structures</td>
<td>7</td>
<td>built-in; programmer-created; dynamic</td>
</tr>
<tr>
<td>Intermediate Human-Computer Interaction</td>
<td>2</td>
<td>universal principles; human-centered considerations; usability testing and verification; design trade-offs; secure user interfaces; standard API graphics</td>
</tr>
<tr>
<td>Course Title</td>
<td>Level</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Intermediate programming constructs</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Object-oriented design and modeling</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Object-oriented programming</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Simple database integration</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Societal and Professional Issues</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Software assurance</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Software development</td>
<td>4</td>
<td></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. 2.
  Discuss significant trends and societal impacts related to computing, software, and the Internet.

**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**
  
  CS. 5.
  Analyze the execution of searching and sorting algorithms.
  CS. 14.
  Construct multiple-file or multiple-module programming solutions that use class hierarchies, inheritance, and polymorphism to reuse existing design and code.
  CS. 15.
  Construct object oriented programming solutions for reuse, using ADTs that incorporate encapsulation, data abstraction, and information hiding.
  CS. 18.
  Create programming solutions that use data structures and existing libraries.
  CS. 23.
  Design and develop secure and fault-tolerant programs that mitigate potential security vulnerabilities.
  CS. 30
  Produce graphical user interfaces that incorporate simple color models and handle events.
  CS. 34.
  Verify program correctness through the development of sound test plans and the implementation of comprehensive test cases.