Computer Science I Metrics

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

Assessment Rubric

**CS. 1. Choose professional behavior in response to ethical issues inherent in computing.**

Emerging Standard
Explains the concepts of intellectual property, plagiarism, and software piracy.

Developed Standard
Chooses to respond professionally to ethical issues in computing, such as intellectual property, plagiarism, and software piracy.

Highly Developed Standard
Values and respects intellectual property, and chooses to act professionally.

**CS. 9. Apply secure coding techniques to object-oriented programming solutions.**

Emerging Standard
Describes secure coding techniques of an object-oriented program, such as public versus private members, data integrity, and data typing.
Developed Standard
Applies secure coding techniques to an object-oriented program.
Highly Developed Standard
Devises a fully secure object-oriented program.

CS. 10. Apply the program development process to problems that are solved using fundamental programming constructs and predefined data structures.

Emerging Standard
Summarizes the phases of the program development cycle.
Developed Standard
With guidance during the design phase, produces working code and performs some testing.
Highly Developed Standard
Develops a working program solution by implementing design, coding, and testing that includes error checking.

CS. 13. Compare and contrast the primitive data types of a programming language; describe how each is stored in memory; and identify the criteria for selection.

Emerging Standard
Names the built-in data types of the programming language.
Developed Standard
Differentiates among the built-in data types and explain when it is appropriate to choose one over another.
Highly Developed Standard
Consistently produces programming solutions with the correct data types implemented.

CS. 19. Decompose a program into subtasks and use parameter passing to exchange information between the subparts.

Emerging Standard
With guidance translates a problem into a programming solution with subtasks.
Developed Standard
With guidance for program analysis and design, decomposes a problem into program components that share data.
Highly Developed Standard
Independently analyzes a problem, formulates a design strategy, and decomposes a problem into program components that share data.

CS. 20. Describe the language translation phases of compiling, interpreting, linking and executing, and differentiate the error conditions associated with each phase.

Emerging Standard
Defines the programming language terms of compiling, interpreting, linking, executing, and error conditions.
Developed Standard
Describes the programming language translation phases of compiling, interpreting, linking, and executing, and differentiates the error conditions associated with each phase.

Highly Developed Standard
Compares the programming language translation phases of compiling, interpreting, linking, and executing, and distinguishes the error conditions associated with each.

**CS. 24. Differentiate between the object-oriented, structured, and functional programming methodologies.**

Emerging Standard
Recognizes the differences and similarities of the object-oriented, structured, and functional programming methodologies.

Developed Standard
Differentiates between the object-oriented, structured, and functional programming methodologies.

Highly Developed Standard
Compares and contrasts the three prominent methodologies of object-oriented, structured, and functional programming.

**CS. 29. Produce algorithms for solving simple problems and trace the execution of computer programs.**

Emerging Standard
Defines the steps necessary to solve a programming problem.

Developed Standard
Produces a working programming solution for a given algorithm.

Highly Developed Standard
Develops a generic solution for an algorithm that can be used to solve a range of related problems.

Association for Computing Machinery

- [Login](#)
- [Acceptable Use](#)
- [Privacy Policy](#)

Copyright © 2021 ACM CCECC