Software Engineering Program Objectives

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. 4. Analyze the efficiency of recursive algorithms.
  CS. 5. Analyze the execution of searching and sorting algorithms.
  CS. 6. Apply fundamental theorems and rules of Calculus to differentiate and integrate algebraic, trigonometric, inverse trigonometric and transcendental functions.
  CS. 7. Apply fundamental theorems and rules of Calculus to evaluate limits and analyze the continuity of various functions.
  CS. 8. Apply fundamental theorems and rules of differentiation to solve problems that model real-world situations.
  SE. 1. Apply mathematical induction and other techniques to prove mathematical results.
  CS. 9. Apply secure coding techniques to object-oriented programming solutions.
  CS. 10. Apply the program development process to problems that are solved using fundamental programming constructs and predefined data structures.
  CS. 11. Assess the appropriateness of using recursion to solve a given problem.
  CS. 12. Compare and contrast a range of searching and sorting algorithms and analyze time and space efficiencies.
  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.
  SE. 2. Construct a preliminary investigative report for a proposed system that includes scheduling and plans for mitigating potential risks.
  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. 16. Construct symbolic models of applied problems described in words.
  CS. 17. Create effective, efficient and secure software, reflecting standard principles of software engineering and software assurance.
  CS. 18. Create programming solutions that use data structures and existing libraries.
  CS. 19. Decompose a program into subtasks and use parameter passing to exchange information between the subparts.
  SE. 3. Decompose complex systems using best practice object-oriented analysis and design tools and techniques.
CS. 20. Describe the language translation phases of compiling, interpreting, linking and executing, and differentiate the error conditions associated with each phase.
CS. 21. Design and construct programming solutions using a variety of recursive techniques.
CS. 22. Design and develop reusable software using appropriate data structures and templates.
CS. 23. Design and develop secure and fault-tolerant programs that mitigate potential security vulnerabilities.
SE. 7. Design and implement consistent and coherent user-centered interfaces that comply with UI standard practices.
CS. 24. Differentiate between the object-oriented, structured, and functional programming methodologies.
SE. 4. Evaluate and test software system requirements that considers both validation and verification.
CS. 25. Examine the logical validity of arguments and proofs as they apply to Boolean expressions.
CS. 26. Illustrate the basic terminology and properties of graphs and trees.
CS. 27. Perform binary and hexadecimal conversions of numbers.
CS. 28. Perform computations using recursively defined functions and structures.
CS. 29. Produce algorithms for solving simple problems and trace the execution of computer programs.
CS. 30. Produce graphical user interfaces that incorporate simple color models and handle events.
SE. 5. Solve problems involving sets, relations, functions, and congruences.
CS. 31. Use fundamental concepts of Calculus to construct graphs of polynomial, rational and exponential functions.
CS. 32. Use graphs and trees to solve problems algorithmically.
SE. 6. Use methods of combinatorics to solve counting problems.
CS. 33. Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software application that is fully implemented and formally tested.
CS. 34. Verify program correctness through the development of sound test plans and the implementation of comprehensive test cases.

Ethics
An ability to exhibit professional, legal, and ethical behavior.

Communication
An ability to read and interpret technical information, as well as listen effectively to, communicate orally with, and write clearly for a wide range of audiences.

Computing and Mathematics
An ability to apply knowledge of computing and mathematics appropriate to the discipline.

Scientific Method
An ability to interpret data, think critically and apply the scientific method.
Society, Privacy, and Responsibility
An ability to demonstrate social awareness, respect for privacy and responsible conduct.

Algorithms
An ability to analyze a problem and craft an appropriate algorithmic and/or engineering solution.

Computer Systems
An ability to design a secure system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability.

Global Impact
An ability to analyze the global impact of software solutions on individuals, organizations, and society.

Team Member
An ability to function effectively as a member of a multidisciplinary team to accomplish common goals.

Association for Computing Machinery
- Login
- Acceptable Use
- Privacy Policy

Copyright © 2019 ACM CCECC