Engineering Applications of Computer Science Principles

PEIMS Code: N1303772

Abbreviation: EACSP

Grade Level(s): 9-12

Award of Credit: 1.0

Approved Innovative Course

- Districts must have local board approval to implement innovative courses.
- In accordance with Texas Administrative Code (TAC) §74.27, school districts must provide instruction in all essential knowledge and skills identified in this innovative course.
- Innovative courses may only satisfy elective credit toward graduation requirements.
- Please refer to TAC §74.13 for guidance on endorsements.

Course Description:

Engineering Applications of Computer Science Principles (EACSP) engages students in computational thinking and design thinking to solve human-centered, technically challenging projects at the intersection of engineering and computer science. Projects are scaffolded to build both technical skills (e.g., Python programming, engineering design) and employability skills (e.g., communication, collaboration, problem solving). Students progress from reverse engineering and improving existing code for a custom photo filter to designing a video-based physical therapy feedback system to creating a camera-controlled wheelchair to assist people with paralysis. All units provide clear core learning requirements with optional extensions, making EACSP suitable, relevant, and rigorous for students in grades 9-12 regardless of previous programming experience. By engaging learners in meaningful versions of the practices of professional engineers and computer scientists, this course sparks a passion for engineering, computational thinking, and problem solving while teaching skills that are foundational to careers in Texas' rapidly growing professional, scientific, and technical services industry.

Essential Knowledge and Skills:

(a) General requirements. This course is recommended for students in grades 9-12. Prerequisite: Algebra 1. Students shall be awarded one credit for successful completion of this course.

(b) Introduction.

(1) Career and technical education instruction provides content aligned with challenging academic standards and relevant technical knowledge and skills for students to further their education and succeed in current or emerging professions.

(2) The Science, Technology, Engineering, and Mathematics (STEM) Career Cluster focuses on planning, managing, and providing scientific research and professional and technical services, including laboratory and testing services, and research and development services.

(3) *Engineering Applications of Computer Science Principles* teaches rigorous engineering design practices, engineering habits of mind, and the foundational tools of computer science. Students apply core computer science principles to solve engineering design challenges that cannot be solved without such knowledge and skills. Students use a variety of computer software and hardware applications to complete projects.

(4) Students are encouraged to participate in extended learning experiences such as career and technical student organizations and other leadership or extracurricular organizations.

(5) Statements that contain the word "including" reference content that must be mastered, while those containing the phrase "such as" are intended as possible illustrative examples.

(c) Knowledge and skills.

(1) The student demonstrates professional standards/employability skills as required by business and industry. The student is expected to:

(A) cooperate, contribute, and collaborate as a member of a group in an effort to achieve a positive collective outcome;

(B) present written and oral communication in a clear, concise, and effective manner;

(C) demonstrate time-management skills in prioritizing tasks, following schedules, and performing goal-relevant activities in a way that produces efficient results;

(D) identify and complete tasks with the highest standards to ensure quality products and services; and

(E) analyze cost savings by using a simulation to run experiments before committing more resources.

(2) The student applies concepts of critical thinking and problem solving. The student is expected to:

(A) analyze elements of an engineering problem to develop creative and innovative solutions;

(B) analyze the elements and structure of a programming problem to develop creative and innovative solutions;

(C) examine information from a customer and existing program to identify pertinent information for the problem-solving task;

(D) compare alternatives using a variety of problem-solving and critical-thinking skills; and

(E) conduct technical research to gather information necessary for decision making.

(3) The student conducts computer science and engineering laboratory activities using safe and environmentally appropriate practices. The student is expected to:

(A) identify and demonstrate safe practices during hands-on cutting and building activities;

(B) identify and demonstrate safe use and storage of electrical components; and

(C) identify and demonstrate and apply appropriate use and conservation of resources, including disposal, reuse, or recycling of materials.

(4) The student applies ethical considerations in designing solutions. The student is expected to:

(A) define and evaluate constraints pertaining to a problem;

(B) incorporate safety considerations with respect to the system, engineer, and user; and

(C) investigate and explain the importance of relevant legal and ethical concepts in computer science such as intellectual property, use of open-source software, attribution, patents, and trademarks.

(5) The student demonstrates an understanding of the structured methods used to collect and analyze information about customer needs. The student is expected to:

(A) analyze all information provided by the customer;

(B) describe joint angle conventions;

(C) create a process flow diagram based on customer needs to generate ideas for potential user actions, product functions, and design opportunities;

(D) develop a flowchart for a program using the results of a process flow diagram;

(E) create a target specifications table;

(F) describe similar existing solutions; and

(G) construct a functional model based on customer needs to generate ideas for potential user actions, product functions, and design opportunities.

(6) The student develops a user interface and thorough but concise supplemental instructions. The student is expected to:

(A) identify the essential tasks to be done by the user;

(B) identify points of potential confusion or unexpected input by the user;

(C) design a software or user interface that clearly communicates to the user how to achieve desired tasks;

(D) develop supplemental user instructions to inform the user of items that cannot be incorporated into the interface, such as how to start the program or frequently asked questions;

(E) test the instructions and program with a student who is not familiar with the project;

(F) evaluate the feedback and results from new user testing;

(G) improve and refine the instructions and program based on feedback and results of testing; and

(H) re-test the instructions and program as necessary.

(7) The student systematically reverse engineers a product, examines ways to improve it, and identifies the type of redesign required to make that improvement. The student is expected to:

(A) test and try to “break” an existing program to determine functionality;

(B) describe unexpected findings from deconstructing existing code;

(C) examine relevant software libraries to determine their uses and functionality;
(D) construct a flowchart for an existing program;
(E) compare the program's current functionality to the customer's needs;
(F) add to the flowchart to meet customer needs;
(G) develop new code to achieve all desired outcomes; and
(H) compare the predicted versus actual functionality of the product to generate ideas for redesign.

(8) The student applies concept generation and selection skills. The student is expected to:
(A) create a black box and functional model of the system;
(B) implement brainstorming, mind mapping, concept sketching, and gallery walk activities to produce new ideas; and
(C) apply concept selection techniques such as a Pugh chart or a weighted decision matrix.

(9) The student develops and applies other engineering skills. The student is expected to:
(A) select and use appropriate tools and techniques to support design activities;
(B) report information about software design solutions in an engineering notebook;
(C) develop, test, and refine programming concepts throughout the development process;
(D) interpret an electrical diagram and use it to build a circuit;
(E) create a circuit using a microcontroller, a breadboard, and multiple components;
(F) apply the design process from different starting points by beginning with a baseline design;
(G) use a model or simulation which represents phenomena and mimics real-world events to develop and test hardware;
(H) evaluate the simulator's strengths and weaknesses for use in improving rocket performance;
(I) critique the usefulness and limitations of certain models;
(J) develop a prototype solution, test the prototype solution against requirements, constraints, and specifications, and refine the prototype solution; and
(K) report and describe the finalized design.

(10) The student applies mathematics and algorithms in programs. The student is expected to:
(A) apply mathematical concepts from algebra, geometry, trigonometry, and calculus to calculate the angle of a joint;
(B) apply mathematical calculations cyclically in a program using algorithms; and
(C) evaluate and verify algorithms for appropriateness and efficiency.

(11) The student develops computer programs to support design solutions. The student is expected to:
(A) design software interfaces that communicate with hardware;

(B) identify and apply relevant concepts from computer science, science, and mathematics such as functions, electricity, and mechanics; and

(C) employ abstraction in a program by representing numerical sensor readouts distance and brightness ranges in more intuitive variables and functions.

(12) The student develops and applies other computer science skills. The student is expected to:

(A) apply systems-thinking skills, emphasizing the integration of small discrete programs into a larger complete program solution;

(B) use intuitive variable names and add comments to code to improve readability;

(C) employ abstraction in a program by representing images as data arrays and representing numerical tone frequencies as variables;

(D) convert image information into the correct data type necessary for given library functions;

(E) develop an algorithm that includes logic, such as “while” and “if”, to accept user trackbar input and display image changes in real time;

(F) document software design solutions through developing flowcharts, pseudocode, and commented code;

(G) design software interfaces that communicate with users and hardware;

(H) employ abstraction to program to an interface, treating imported code as a "black box;"

(I) employ abstraction by representing a joint as four points in a plane; and

(J) apply a well-defined programming vocabulary and skill set.

(13) The student develops and uses computer programs to process data and information to gain insight and discover connections to support design solutions. The student is expected to:

(A) organize complex image and video data appropriately for processing;

(B) analyze complex data to make decisions and instruct users; and

(C) develop programs that use incoming data and algorithms to create output data, information, and commands.

Recommended Resources and Materials:

This course requires student access to the following:

- personal computer (either laptop or desktop)
- webcam
- basic engineering lab equipment (*e.g.*, tools, safety goggles)
- single-board computers (*e.g.*, Raspberry Pis)
- free software for programming (*e.g.*, a Python integrated development environment)
- free libraries of existing Python code (*e.g.*, OpenCV)

A full set of course equipment and supplies is available for purchase as a kit from studica.com (https://www.studica.com/engineering-applications-of-computer-science-course); alternatively,

schools may elect to source materials separately from multiple vendors using an equipment and supplies list provided by *Engineer Your World*.

Teaching materials for this course include detailed instructional materials and sample solution code from *Engineer Your World*. A sample of the instructional materials provided by *EYW* is available for review upon request through engineeryourworld.org.

Supplementary outside supporting resources include the following:

Matthes, Eric. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019.

"Python For Beginners." Python. Accessed March 1, 2023. https://www.python.org/about/gettingstarted/.

"OpenCV Tutorials." OpenCV. Last modified December 29, 2022. https://docs.opencv.org/4.x/d9/df8/tutorial_root.html.

"NumPy: the absolute basics for beginners." NumPy. Accessed March 1, 2023. https://numpy.org/doc/stable/user/absolute_beginners.html.

Carnes, Beau. "Python for Beginners – Full Course [Programming Tutorial]." freeCodeCamp. August 9, 2022. Video, 4:39:59. https://www.youtube.com/watch?v=eWRfhZUzrAc.

Recommended Course Activities:

EACSP engages students in computational thinking and design thinking to solve human-centered, technically challenging projects at the intersection of engineering and computer science. Projects are scaffolded to build both technical skills (*e.g.*, Python programming, engineering design) and employability skills (*e.g.*, communication, collaboration, problem solving.) Each of the course's five major projects builds skills that are required for successful completion of the subsequent projects.

Unit 1: Photo-and-Text Memes (Programming in Python) teaches students the fundamental skills required to create computer code in the Python programming language that they will use throughout the course. Students develop basic coding skills while creating a program that allows a user to combine a picture with overlaid text, essentially creating their own memes. Students explore the style and syntax of Python code while learning and practicing skills that are foundational to the course.

Unit 2: Custom Photo Filters (Exploring Images) invites students to create their own digital image processing programs to convert a student-selected color or grayscale image to a six-tone, Warhol-style picture. Students use engineering tools and techniques to "reverse engineer" existing code, model user needs, and develop a project plan for the challenge. They learn to build on existing code, integrate libraries of code, and create a software user interface that will allow an artist to "paint" images by controlling colors with slider bars. Finally, students write instructions for the artist customer who will use the program. Students' coding and writing skills are put to the test when the

program and its documentation are handed over to students in another class, such as an art class or *Engineering Design and Analysis (EYW)*, for user testing and feedback.

Unit 3: Computer-Assisted Physical Therapy Tools (Analyzing Video) engages students in engineering design to develop a real-time feedback tool for physical therapy patients performing rehabilitation exercises outside of the clinical setting. After analyzing user needs and creating a functional model for the system, students design and build a basic wearable device that allows a web camera to capture information about a joint's range of motion. Students write programs that analyze large quantities of video data, apply an algorithm for calculating changing joint angles, provide real-time user feedback, and export data to a file for later analysis by the patient's physical therapist. The challenge ends with students developing recommendations for future improvements to their systems.

Unit 4: Mechatronic Assistive Devices (Building and Coding) engages students in building and programming mechatronic devices that incorporate Raspberry Pis (extremely affordable, pocket-sized computers) with structural elements, sensors, motors, lights, and other physical components. Students build and program scale models of assistive devices (*e.g.*, an automated "lazy Susan" to assist people with disabilities, a solar tracking device to maximize efficiency of solar panels, an automated "smart lighting" control system). Once each device is working, students use engineering concept generation and selection techniques to create improvements that enhance functionality. By integrating mechanical redesign with improvements to algorithm and code, students develop devices that better serve their customers' needs.

Unit 5: Camera-Controlled Wheelchair (System Design) engages students in developing a wheelchair control system to improve independence and mobility for people with quadriplegia. Students analyze customer needs and generate design concepts for a physical apparatus to capture user head movements via a chair-mounted camera. They integrate original and off-the-shelf hardware and software to control chair speed and direction based on the detected head motion, create testable prototypes of their designs, and refine their systems through iteration.

(Bonus) Unit 6: Water Rockets (Model Selection and Performance) teaches students to use a computer model to customize a water rocket to reach maximum altitude. Students select an initial rocket design, use the model to predict its maximum altitude, build and launch a physical rocket, compare the actual altitude to the predicted altitude, discuss the accuracy of the model, and make informed revisions to their rockets. Students then predict the performance of their revised design, relaunch, and compare results to predictions.

Suggested methods for evaluating student outcomes:

Assessment tools provided with the EACSP curriculum include individual student content assessments; project rubrics; end-of-unit reports and presentations; peer and self-assessments; and guidelines for code submissions, work product submissions, and notebook checks.

1. **Student content assessments.** The curricular materials for EACSP include individual student content assessments, each of which is tied to a particular student learning objective. Many of these assessments have been co-developed with teachers in EACSP's community of

practice, members of which have worked together to develop and enhance “grading keys" (*e.g.*, sample student responses, grading guidelines) that enable teachers to check for evidence of student understanding of important concepts and learning objectives. Most of these assessments are formative in nature, enabling teachers to adjust instruction and reinforce learning.

2. **Project rubrics.** While sample project rubrics are provided as a reference for teachers, the actual project rubrics used in each class are co-created with students at a strategic point in each unit, usually immediately before teams begin to generate design concepts. (The timing of the rubric development is intended to maximize students’ opportunities to construct their own understanding early in the process while also ensuring that teams generate ideas with the full knowledge of how their designs will be judged.) With ample support from the curriculum, teachers guide their students in breaking each project into sub-categories (*e.g.*, physical device, system description, final program, formal report) and identifying essential components of each sub-category that should be evaluated. The teacher then defines the criteria for obtaining points for each component. This process increases student engagement, buy-in, and ownership of their learning, while also allowing the teacher to determine and demonstrate expectations for quality of work.

3. **End-of-unit report and presentation guidelines.** Guidelines provided in the curriculum are intentionally general so that they may serve as the starting point for a class discussion of what students should include in each final report. As with the project rubric, teachers are encouraged to engage students in creating the report rubric against which their final report will be assessed.

4. **Peer assessments and self-assessments.** Peer assessments allow students to provide their team members with feedback on their contributions to overall team performance. Self-assessments are crafted to encourage students to reflect on the feedback from their peers and to identify opportunities for growth. These instruments are typically administered at the end of each unit, although they may also be used during a unit to help “unstick” a struggling team.

5. **Routine code submissions, work product submissions, and notebook checks.** The curriculum calls out multiple points in each unit where the teacher should check students’ code, notebook documentation, and other work products. Frequent monitoring of these artifacts helps the teacher ensure that teams are on track and provides opportunities to give formative assessments and feedback.

Teacher qualifications:

At least one of the following certificates is recommended:

(1) Master Science Teacher (Grades 8-12),

(2) Mathematics/Physical Science/Engineering: Grades 6-12,

(3) Mathematics/Physical Science/Engineering: Grades 8-12,

(4) Physical Science: Grades 6-12,

(5) Physical Science: Grades 8-12,

(6) Physics/Mathematics: Grades 7-12,

(7) Physics/Mathematics: Grades 8-12,

(9) Science, Technology, Engineering, and Mathematics: Grades 6-12,

(10) Secondary Industrial Arts (Grades 6-12),

(11) Secondary Industrial Technology (Grades 6-12),

(12) Secondary Physics (Grades 6-12),

(13) Technology Education: Grades 6-12

(14) Technology Applications Grades 8-12

(15) Technology Applications EC -12

(16) Computer Science Grades 8-12

(17) Secondary computer information systems Grades 6-12

(18) Junior High School (9-10) or High School Computer Information Systems

(19) Grades 6-12 or Grades 9-12 Computer information Systems.

Additional information:

Teachers appointed by their schools to teach EACSP must first complete a two-week professional development institute (PDI) offered by *Engineer Your World* at The University of Texas at Austin. Developed in alignment with the research-based Standards for Professional Development for Teachers of Engineering (https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1107&context=jpeer), this PDI engages teachers in the same authentic practices that their students will undertake, enabling them to experience the curriculum they will teach and preparing them to meet the unique challenges of their classroom. The cost for the PDI is $2,000 per teacher.