Specialized Platform Development (SPD)

Preamble

The Specialized Platform Development (SPD) Knowledge Area (KA) refers to attributes involving creating a software platform to address specific needs and requirements for particular areas. Specialized platforms, such as healthcare providers, financial institutions, or transportation companies, are tailored to meet specific needs. Developing a specialized platform typically involves several key stages, i.e., requirements, design, development, deployment, and maintenance.

The SPD Beta Versions considered the following factors:

- ***Emerging Computing Areas*** such as data science/analytics – use multi- platforms to retrieve sensing data. Cybersecurity – involves protecting certain data extraction, recognizing protocols to protect network transfer ability, and manipulating it. Artificial intelligence and machine learning – use artifacts that retrieve information for robotics and drones to perform specific tasks. This continuous emergence of computing technology areas has increased the appetite to develop platforms communicating software with specialized environments. This need has also increased the need to develop specialized programming languages for these platforms, such as web and mobile development. The Interactive Computing Platform addresses the advent of Large Language Models (LLMs), such as OpenAI's ChatGPT, OpenAI's Codex, and GitHub's Copilot, in addition to other platforms that perform data analysis, and visualizations.
- **Industry needs and competencies** have created a high demand for developers on specialized platforms, such as mobile, web, robotics, embedded, and interactive. Some of the unique professional competencies obtained by current job descriptions relevant to this KA are:
    - *Create a mobile app that provides a consistent user experience across various devices, screen sizes, and operating systems.*
    - *Analyze people's experience using a novel peripheral for an immersive system facilitated using a head-mounted display and mixed reality, with attention to usability and accessibility specifications.*
    - *Build and optimize a secure web page for evolving business needs using a variety of appropriate programming languages.*
    - *Develop application programming interfaces (APIs) to support mobile functionality and remain current with the terminology, concepts, and best practices for coding mobile apps.*
    - *Availability of devices and artifacts, such as raspberry PIs, Arduinos, and mobile devices. The low cost of microcontrollers and devices, such as robots using ROS that can perform specialized actions*

Changes since CS 2013

The consideration of these factors resulted in the following significant changes from the CS2013 version:

- **Renamed of the Knowledge Area name**: From Platform-Based Development (PBD) to Specialized Platform Development due to the specific needs of the already mentioned tasks. This particular KA is often called *software platform development* since the specialized development takes part in the software stages for multi-platform development.
- **Increase of Computer Science Core Hours**: Based on the already mentioned needs, the SPD beta version has increased the number of computer science course hours from 0 to 9. The KA subsets the web and mobile knowledge units (often the most closely related units in CS Core) into foundations and specialized platforms core hours to provide flexibility and adaptability. This division allows programs at different institutions to offer different interests, concentrations, or degrees that focus on different application areas, where many of these concepts intersect the CS core. Therefore, the *common aspects web* and mobile foundations have concepts in many CS programs' core. Finally, the rest of the knowledge units permit the curriculum to have an extended and flexible number of KA core hours.
- **Renamed old knowledge units and incorporated new ones**: in the spirit of capturing the future technology and societal responsibilities, the Robotics, Embedded Systems, and Society, Ethics, and Professionalism (SEP) knowledge units were introduced in this version. These KU work harmoniously with other KAs, consistent with the topics and concepts covered in specific KAs' knowledge units.

Specialized platform development provides a deep understanding of a particular user group's needs and requirements and considers other knowledge areas that help design and build a platform that meets other KA's needs. Considering other KAs' needs, SPD helps to streamline workflows, improve efficiency, and drive innovation across the recommended curriculum discussed in CS2023.

Core Hours

Knowledge Units	CS Core	KA Core
SPD/Common Aspects	3	*
SPD/Web Platforms		*
SPD/Mobile Platforms		*
SPD/Robot Platforms		*
SPD/Embedded Platforms		*
SPD/Game Platforms		*
SPD/Interactive Computing Platforms		*

Total	**2**	

Knowledge Units

SPD-Common: Common Aspects/Shared Concerns

CS-Core:

1. Overview of development platforms (i.e., web, mobile, game, robotics, embedded, and Interactive)
    a. Input/sensors/control devices/haptic devices
    b. Resource constraints
        i. Computational
        ii. Data storage
        iii. Memory
        iv. Communication
    c. Requirements
        i. security, uptime availability, fault tolerance (See also: SE-Reliability)
    d. Output/actuators/haptic devices
2. Programming via platform-specific Application Programming Interface (API) vs traditional application construction
3. Overview of platform Languages, (e.g., Python, Swift, Lua, Kotlin)
4. Programming under platform constraints and requirements (e.g., available development tools, development, security considerations)
5. Techniques for learning and mastering a platform-specific programming language.

Illustrative Learning Outcomes

1. List the constraints of mobile programming
2. Describe the three-tier model of web programming
3. Describe how the state is maintained in web programming
4. List the characteristics of scripting languages

SPD-Web: Web Platforms

KA-Core:

1. Web programming languages (e.g., HTML5, JavaScript, PHP, CSS)
2. Web platforms, frameworks, or meta-frameworks
    a. Cloud services
    b. API, Web Components
3. Software as a Service (SaaS)
4. Web standards such as document object model, accessibility (See also: HCI-Accessability: 2)
5. Security and Privacy considerations (See also, SEP-Security)

Non-Core:

6. Analyzing requirements for web applications
7. Computing services (see also, DM-NoSQL: 8)
    a. Cloud Hosting
    b. Scalability (e.g., Autoscaling, Clusters)
    c. How to estimate costs for these services (based on requirements or time usage)
8. Data management (See also: DM-Core)
    a. Data residency (where the data is located and what paths can be taken to access it)
    b. Data integrity: guaranteeing data is accessible and guaranteeing that data is deleted when required
9. Architecture
    a. Monoliths vs. Microservices
    b. Micro-frontends
    c. Event-Driven vs. RESTful architectures: advantages and disadvantages
    d. Serverless, cloud computing on demand
10. Storage solutions (See also: DM-Relational/ DM-SQL)
    a. Relational Databases
    b. NoSQL databases

Illustrative Learning Outcomes

1. Design and Implement a web application using microservice architecture design.
2. Describe the web platform's constraints and opportunities, such as hosting, services, and scalability, that developers should consider.
3. Compare and contrast web programming with general-purpose programming.
4. Describe the differences between Software-as-a-Service and traditional software products.
5. Discuss how web standards impact software development.
6. Review an existing web application against a current web standard.

SPD-Mobile: Mobile Platforms

KA-Core:

1. Development
    a. Mobile programming languages
    b. Mobile programming environments
2. Mobile platform constraints
    a. User interface design (See also: HCI: Understanding the User, GIT-Image Processing)
    b. Security
3. Access
    a. Accessing data through APIs (See also: DM-Quering)
    b. Designing API endpoints for mobile apps: pitfalls and design considerations
    c. Network and the Web interfaces (See also: NC-Introduction, DM-Modeling:8.d)

Non-Core:

4. Development
    a. Native versus cross-platform development
    b. Software design/architecture patterns for mobile applications (See also: SE-Design)
5. Mobile platform constraints
    a. Responsive user interface design (see also: HCI-Accessibility and Inclusive Design)
    b. Diversity and mobility of devices
    c. User experiences differences (e.g., between mobile and web-based applications)
    d. Power and performance tradeoff
6. Mobile computing affordances
    a. Location-aware applications
    b. Sensor-driven computing (e.g., gyroscope, accelerometer, health data from a watch)
    c. Telephony and instant messaging
    d. Augmented reality (See also: GIT-Immersion)
7. Specification and testing (See also: SDF: Software Development Practices, SE-Validation)
8. Asynchronous computing (See also: PDC)
    a. Difference from traditional synchronous programming
    b. Handling success via callbacks
    c. Handling errors asynchronously
    d. Testing asynchronous code and typical problems in testing

Illustrative Learning Outcomes

1. Develop a location-aware mobile application with data API integration.
2. Build a sensor-driven mobile application capable of logging data on a remote server.
3. Create a communication appincorporating telephony and instant messaging.
4. Compare and contrast mobile programming with general-purpose programming.
5. Evaluate the pros and cons of native and cross-platform mobile app development.

SPD-Robot: Robot Platforms

Non-Core: (See also: AI-Robo)

1. Types of robotic platforms and devices
2. Sensors, embedded computation, and effectors (actuators) (See also: GIT-Physical:Tangible)
3. Robot-specific languages and libraries
4. Robotic platform constraints and design considerations
5. Interconnections with physical or simulated systems
6. Robotics
    a. Robotic software architecture (e.g., using the Robot Operating System)
    b. Forward kinematics (See also: GIT-Animation)
    c. Inverse kinematics (See also: GIT-Animation)
    d. Dynamics

e. Navigation and robotic path planning (See also: AI-Robo)
f. Manipulation and grasping (See also: AI-Robo)
g. Safety considerations (See also: SEP-Professional, SEP-Context) // move to SEP

Illustrative Learning Outcomes

1. Design and implement an application on a given robotic platform
2. Assemble an Arduino-based robot kit and program it to navigate a maze
3. Compare robot-specific languages and techniques with those used for general-purpose software development
4. Explain the rationale behind the design of the robotic platform and its interconnections with physical or simulated systems,
5. Given a high-level application, design a robot software architecture using ROS specifying all components and interconnections (ROS topics) to accomplish that application
6. Discuss the constraints a given robotic platform imposes on developers

SPD-Embedded: Embedded Platforms

This Knowledge unit considers embedded computing platforms and their applications. Embedded platforms cover knowledge ranging from sensor technology to ubiquitous computing applications.

Non-Core:

1. Introduction to the unique characteristics of embedded systems
    a. real-time vs. soft real-time and non-real-time systems
    b. Resource constraints, such as memory profiles, deadlines (See also: AR-Memory: 2)
2. API for custom architectures
    a. GPU technology (See also: AR-Heterogeneity: 1, GIT-Interfaces (AI))
    b. Field Programmable Gate Arrays (See also: AR-Logic: 2)
    c. Cross platform systems
3. Embedded Systems
    a. Microcontrollers
    b. Interrupts and feedback
    c. Interrupt handlers in high-level languages (See also: SF-Overview: 5)
    d. Hard and soft interrupts and trap-exits (See also: OS-Principles: 6)
    e. Interacting with hardware, actuators, and sensors
    f. Energy efficiency
    g. Loosely timed coding and synchronization
    h. Software adapters
4. Real-time systems
    a. Hard real-time systems vs. soft real-time systems (See also: OS-Real-time: 3)
    b. Timeliness
    c. Time synchronization/scheduling
    d. Prioritization
    e. Latency
    f. Compute jitter

5. Memory management
    a. Mapping programming construct (variable) to a memory location(See also: AR-Memory)
    b. Shared memory (See also: OS-Memory)
    c. Manual memory management
    d. Garbage collection (See also: FPL-Language Translation)
6. Safety considerations and safety analysis (See also: SEP-Context, SEP-Professional)
7. Sensors and actuators
8. Embedded programming
9. Real-time resource management
10. Analysis and verification
11. Application design

Illustrative Learning Outcomes

1. Design and implement a small embedded system for a given platform (e.g., a smart alarm clock or a drone)
2. Describe the unique characteristics of embedded systems versus other systems
3. Interface with sensors/actuators
4. Debug a problem with an existing embedded platform
5. Identify different types of embedded architectures
6. Evaluate which architecture is best for a given set of requirements
7. Design and develop software to interact with and control hardware
8. Design methods for real-time systems
9. Evaluate real-time scheduling and schedulability analysis
10. Evaluate formal specification and verification of timing constraints and properties

SPD-Game: Game Platforms

The Game Platforms knowledge unit draws attention to concepts related to the engineering of performant real-time interactive software on constrained computing platforms. Material on requirements, design thinking, quality assurance, and compliance enhances problem-solving skills and creativity.

Non-Core:

1. Historical and contemporary platforms for games (See also: AR-A: Digital Logic and Digital Systems, (See also: AR-A: Interfacing and Communication)
    a. *Evolution of Game Platforms*

(e.g., Brown Box to Metaverse and beyond; Improvement in Computing Architectures (CPU and GPU); Platform Convergence and Mobility)

b. *Typical Game Platforms*
(e.g., Personal Computer; Home Console; Handheld Console; Arcade Machine; Interactive Television; Mobile Phone; Tablet; Integrated Head-Mounted Display; Immersive Installations and Simulators; Internet of Things enabled Devices; CAVE Systems; Web Browsers; Cloud-based Streaming Systems)

c. *Characteristics and Constraints of Different Game Platforms*
(e.g., Features (local storage, internetworking, peripherals); Run-time performance (GPU/CPU frequency, number of cores); Chipsets (physics processing units, vector co-processors); Expansion Bandwidth (PCIe); Network throughput (Ethernet); Memory types and capacities (DDR/GDDR); Maximum stack depth; Power consumption; Thermal design; Endian)

d. *Typical Sensors, Controllers, and Actuators* (See also: GIT-Interaction)
(e.g., typical control system designs—peripherals (mouse, keypad, joystick), game controllers, wearables, interactive surfaces; electronics and bespoke hardware; computer vision, inside-out tracking, and outside-in tracking; IoT-enabled electronics and i/o.

e. *Esports Ecosystems*
(e.g., evolution of gameplay across platforms; games and esports; game events such as LAN/arcade tournaments and international events such the Olympic Esports Series; streamed media and spectatorship; multimedia technologies and broadcast management; professional play; data and machine learning for coaching and training)

2. Real-time Simulation and Rendering Systems

a. *CPU and GPU architectures*: (See also, AR-Heteroginity: 2)
(e.g., Flynn's taxonomy; parallelization; instruction sets; common components—graphics compute array, graphics memory controller, video graphics array basic input/output system; bus interface; power management unit; video processing unit; display interface)

b. *Pipelines for physical simulations and graphical rendering:* (See also, GIT-Rendering)
(e.g., tile-based, immediate-mode)

c. *Common Contexts for Algorithms, Data Structures, and Mathematical Functions*: (See also, MSF:?, AL-Fundamentals)
(e.g., game loops; spatial partitioning, viewport culling, and level of detail; collision detection and resolution; physical simulation; behavior for intelligent agents; procedural content generation)

d. *Media representations* (See also, GIT-Fundamentals: 8, GIT-Geometric: 3)
(e.g., i/o, and computation techniques for virtual worlds: audio; music; sprites; models and textures; text; dialogue; multimedia (e.g., olfaction, tactile).

3. Game Development Tools and Techniques

a. *Programming Languages*:
(e.g., C++; C#; Lua; Python; JavaScript)

b. *Shader Languages*: HLSL; GLSL; ShaderGraph.
c. *Graphics Libraries and APIs* (See also, GIT-Rendering, HCI-System Design: 3)
(e.g., DirectX; SDL; OpenGL; Metal; Vulkan; WebGL)
d. *Common Development Tools and Environments*: (See also: SDF-Practices: 2, SE-Tools,)
(e.g., IDEs; Debuggers; Profilers; Version Control Systems including those handling binary assets; Development Kits and Production/Consumer Kits; Emulators)

4. Game Engines
   a. Open Game Engines
   (e.g.,Unreal; Unity; Godot; CryEngine; Phyre; Source 2; Pygame and Ren'Py; Phaser; Twine; SpringRTS)
   b. *Techniques (See also: AR-Performance and Energy Efficiency, (See also: SE-Requirements)*
   (e.g., Ideation; Prototyping; Iterative Design and Implementation; Compiling Executable Builds; Development Operations and Quality Assurance—Play Testing and Technical Testing; Profiling; Optimization; Porting; Internationalization and Localization; Networking)
5. Game Design
   a. *Vocabulary*
   (e.g., game definitions; mechanics-dynamics-aesthetics model; industry terminology; experience design; models of experience and emotion)
   b. *Design Thinking and User-Centered Experience Design* (See also: SE-?)
   (e.g., methods of designing games; iteration, incrementing, and the double-diamond; phases of pre- and post-production; quality assurance, including alpha and beta testing; stakeholder and customer involvement; community management)
   c. *Genres*
   (e.g., adventure; walking simulator; first-person shooter; real-time strategy; multiplayer online battle arena (MOBA); role-playing game (rpg))
   d. *Audiences and Player Taxonomies (See also: HCI-Understanding the User: 5)*
   (e.g., people who play games; diversity and broadening participation; pleasures, player types, and preferences; Bartle, yee)
   e. *Proliferation of digital game technologies to domains beyond entertainment*: (See also: AI-E: Applications and Societal Impact)
   (e.g., Education and Training; Serious Games; Virtual Production; Esports; Gamification; Immersive Experience Design; Creative Industry Practice; Artistic Practice; Procedural Rhetoric.)

Illustrative Learning Outcomes

1. Recall the characteristics of common general-purpose graphics processing architectures
2. Identify the key stages of the immediate-mode rendering pipeline
3. Describe the key constraints a given game platform will likely impose on developers
4. Explain how esports are streamed to large audiences over the Internet
5. Translate complex mathematical functions into performant source code

6. Use an industry-standard graphics API to render a 3D model in a virtual scene
7. Modify a shader to change a visual effect according to stated requirements
8. Implement a game for a particular platform according to specification
9. Optimize a function for processing collision detection in a simulated environment
10. Assess a game's run-time and memory performance using an industry-standard tool and development environment
11. Compare the interfaces of different game platforms, highlighting their respective implications for human-computer interaction
12. Recommend an appropriate set of development tools and techniques for implementing a game of a particular genre for a given platform
13. Discuss the key challenges in making a digital game that is cross-platform compatible
14. Suggest how game developers can enhance the accessibility of a game interface
15. Create novel forms of gameplay using frontier game platforms

SPD-Interactive: Interactive Computing Platforms

Non-Core:

1. Data Analysis Platforms
    a. Jupyter notebooks; Google Colab; R; SPSS; Observable
    b. Cloud SQL/data analysis platforms (e.g., BigQuery)(See also: DM-Quering)
        i. Apache Spark
2. Data Visualizations (See also: GIT:Visualization)
    a. Interactive presentations backed by data
    b. Design tools requiring low-latency feedback loops
        i. rendering tools
        ii. graphic design tools
3. Programming by Prompt
    a. Generative AI (e.g., OpenAI's ChatGPT, OpenAI's Codex, GitHub's Copilot) and LLMs are accessed/interacted.
    b. Quantum Platforms (See also: AR-Quantum, FPL-Quantum: Quantum Computing)
        i. Program quantum logic operators in quantum machines
        ii. Use API for available quantum services
        iii. Signal analysis / Fourier analysis / Signal processing (for music composition, audio/RF analysis) (See also: GIT-Image)

Illustrative Learning Outcomes

1. Interactively analyze large datasets
2. Create a backing track for a musical performance (e.g., with live coding)
3. Create compelling computational notebooks that construct a narrative for a given journalistic goal/story.
4. Implement interactive code that uses a dataset and generates exploratory graphics
5. Create a program that performs a task using LLM systems

6. Contrast a program developed by an AI platform and by a human
7. Implement a system that interacts with a human without using a screen
8. Contextualize the attributes of different data analysis styles, such as interactive vs. engineered pipeline
9. Write a program using a notebook computing platform (e.g., searching, sorting, or graph manipulation)
10. Demonstrate a quantum gate outcome using a quantum platform

SPD-SEP/Mobile

Topics

1. Privacy and data protection
2. Accessibility in mobile design
3. Security and cybersecurity:
4. Social impacts of mobile technology
5. Ethical use of AI and algorithms

Illustrative Learning Outcomes

1. Understand and uphold ethical responsibilities for safeguarding user privacy and data protection in mobile applications.
2. Design mobile applications with accessibility in mind, ensuring effective use by people with disabilities.
3. Demonstrate proficiency in secure coding practices to mitigate risks associated with various security threats in mobile development.
4. Analyze the broader social impacts of mobile technology, including its influence on communication patterns, relationships, and mental health.
5. Comprehend the ethical considerations related to the use of AI in mobile applications, ensuring algorithms are unbiased and fair.

SPD-SEP/Web

Topics

1.

Illustrative Learning Outcomes

1. Understand how mobile computing impacts communications and the flow of information within society
2. Design mobile apps that have made daily tasks easier/faster
3. Recognize how the ubiquity of mobile computing has affected work-life balance
4. Understand how mobile computing impacts health monitoring and healthcare services
5. Comprehend how mobile apps are used to educate on and help achieve UN sustainability goals

SPD-SEP/Game

Topics

1. Intellectual Property Rights in Creative Industries:

    a. *Intellectual Property Ownership*: copyright; trademark; design right; patent; trade secret; civil versus criminal law; international agreements; procedural content generation and the implications of generative artificial intelligence.
    b. *Licencing*: usage and fair usage exceptions; open-source licence agreements; propietary and bespoke licencing; enforcement.
2. Fair Access to Play:
    a. *Game Interface Usability*: user requirements; affordances; ergonomic design; user research; experience measurement; heuristic evaluation methods for games.
    b. *Game Interface Accessibility:* forms of impairment and disability; means to facilitate access to games; universal design; legislated requirements for game platforms; compliance evaluation; challenging game mechanics and access.
3. Game-Related Health and Safety:
    a. *Injuries in Play:* ways of mitigating common upper body injuries, such as repetitive strain injury; exercise psychology and physiotherapy in esports.
    b. *Risk Assessment for Events and Manufacturing:* control of substances hazardous to health (COSHH); fire safety; electrical and electronics safety; risk assessment for games and game events; risk assessment for manufacturing.
    c. *Mental Health*: motivation to play; gamification and gameful design; game psychology—internet gaming disorder.
4. Platform Hardware Supply Chain and Sustainability:
    a. *Platform Lifecycle*: platform composition—materials, assembly; mineral excavation and processing; power usage; recycling; planned obsolescence.
    b. *Modern Slavery*: supply-chains; forced labour and civil rights; working conditions; detection and remission; certification bodies and charitiable endeavours.
5. *Representation in the Media and Industry:*
    a. *Inclusion*: identity and identification; exclusion of characters diverse audiences identify with; media representation and its effects; media literacy; content analysis; stereotyping; sexualization.
    b. *Equality*: histories and controversies, such as gamergate; quality of life in the industry; professional discourse and conduct in business contexts; pathways to game development careers; social mobility; experience of developers from different backgrounds and identities; gender, and technology.

Illustrative Learning Outcomes

1. Recall the ways in which creators can protect their intellectual property
2. Identify common pitfalls in game interfaces that exclude players with impaired or non-functional vision
3. Describe how heuristic evaluation can be used to identify usability problems in game interfaces
4. Explain why upper body injuries are common in esports
5. Reform characters and dialogues in a scene to reduce stereotype threat
6. Illustrate the way in which the portrayal of race in a game can influence the risk of social exclusion in the associated online community around the game
7. Modify a policy for a LAN party event to include mitigations that lower the risk of fire

8. Design a gamification strategy to motivate serious play for an awareness-raising game
9. Analyse the role of company hiring policies and advocacy on social mobility
10. Assess the appropriateness of two manufacturers for producing a new games console
11. Compare options for open-source licencing of a game development tool
12. Recommend changes to a game interface to improve access to players who are deaf or whose hearing is otherwise impaired
13. Discuss whether games are addictive in nature
14. Suggest how the portrayal of women in video games influences the way players may perceive members of those groups
15. Create a videogame that successfully advocates for climate science

SPD-SEP/Robotics

SPD-SEP/Interactive

This knowledge unit captures the society, ethics, and professionalism aspects from the specialized platform development viewpoint. Every stage from the software development perspective impacts the SEP knowledge unit.

Topics

1. Augmented technology and societal impact
2. Robotic design
3. Graphical User Interfaces Considerations for DEI
4. Recognizing data privacy and implications
5. LLMs and global compliance regulations, such as copyright law
6. Mobile development and global equality

Professional Dispositions

- Learning to learn (new platforms, languages)
- Inventiveness (in designing software architecture within non-traditional constraints)
- Adaptability (to new constraints)

Math Requirements

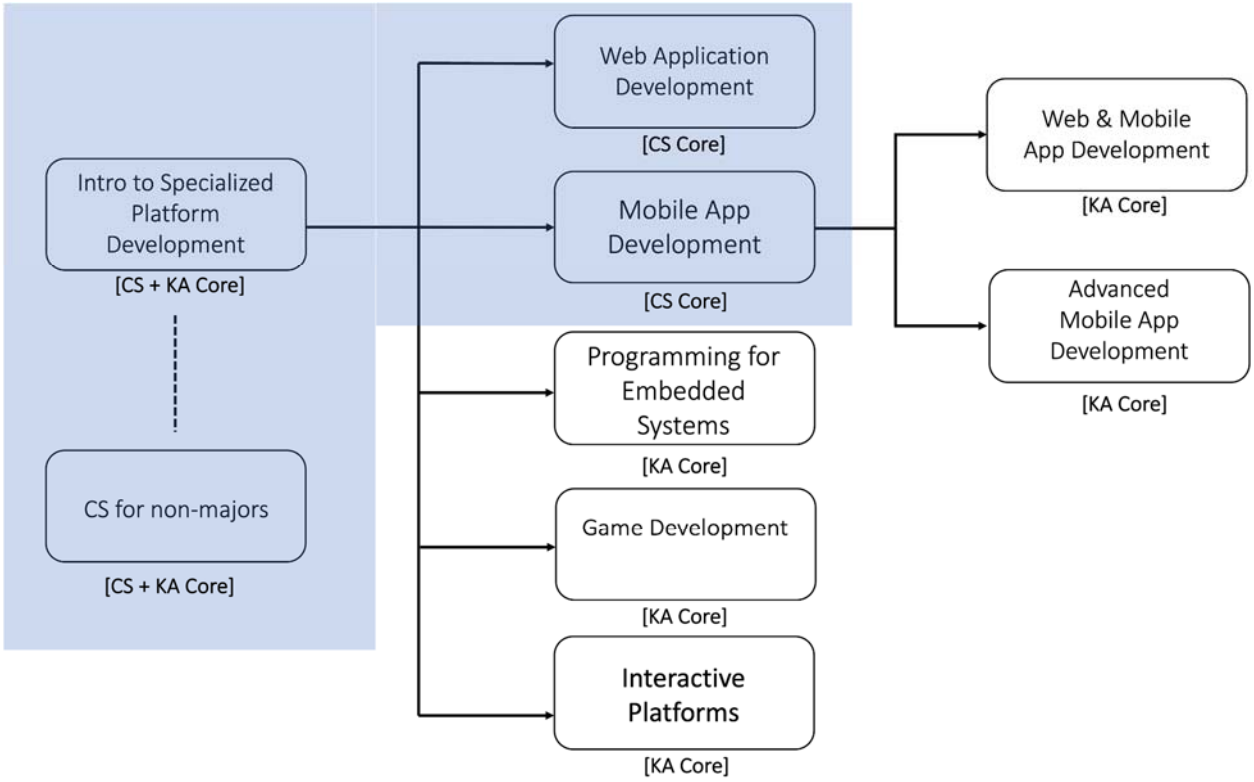
Required:

Desired:

- Equations and Basic Algebra
- Geometry (e.g., 2d and 3d coordinate systems—cartesian and polar—points, lines, and angles)
- Trigonometry

- Vectors, Matrices, and Quaternions—linear transformations, affine transformations, perspective projections, exponential maps, rotation, etc.
- Geometric primitives
- Rigid body kinematics and Newtonian physics
- Signal processing
- Coordinate-space transformations
- Parametric curves
- Binary and Hexadecimal Number Systems
- Logic and Boolean Algebra
- Calculus
- Linear Algebra
- Probability/Statistics (e.g., dynamic systems, visualization e.g., algorithmically generated Tuftian-style displays)
- Discrete Math/Structures (e.g., graphs for process control and path search)

Course Packaging Suggestions



Courses Common to CS and KA Core Approaches
Specialized Platform Development

- **SPD-Common**
- **SPD-Web**
- **SPD-Mobile**
- **SPD-Robotic**

- **SPD-Interactive**

CS for Non-Majors
- **SPD-Common**
-

Mobile Development 8 Week Course
- SPD-Mobile
    - API Design and Development
    - User-Centered Design and the Mobile Platform
    - Software Engineering Applications in Mobile Computing (covers design patterns, testing, async programming, and the like in a mobile context)
    - Challenges with Mobile Computing Security
    - Societal Impact of Mobile Computing
    - Mobile Computing Capstone Course

Committee

Chair: Christian Servin (El Paso Community College, El Paso, TX, USA)

Members:
- Sherif G. Aly, The American University in Cairo, Egypt
- Yoonsik Cheon, The University of Texas at El Paso, El Paso, Texas, USA
- Eric Eaton, University of Pennsylvania, Philadelphia, PA, USA
- Claudia L. Guevara, Jochen Schweizer mydays Holding GmbH, Munich, Germany
- Larry Heimann, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
- Amruth N. Kumar, Ramapo College of New Jersey, Mahwah, NJ, USA
- R. Tyler Pirtle, Google
- Michael James Scott, Falmouth University, UK

Contributors:
- Sean R. Piotrowski, Rider University, USA
- Mark 0'Neil, Blackboard Inc., USA
- John DiGennaro, Qwickly
- Rory K. Summerley, London South Bank University, UK.