Instructional Framework

Software and App Design

11.0202.00

This Instructional Framework identifies, explains, and expands the content of the standards/measurement criteria, and, as well, guides the development of multiple-choice items for the Technical Skills Assessment. This document corresponds with the Technical Standards endorsed on May 14, 2024.

Domain 1: Coding/Programming **Instructional Time: 45 - 55%**	
STANDARD 3.0 UTILIZE PRIMITIVE DATA TYPES AND STRINGS IN WRITING PROGRAMS	

3.1 Differentiate among numeric, Boolean, character, string variables, and float and double	• Various data types
	  ○ Boolean
	  ○ Integers
	  ○ Float/Double
	  ○ Character
	  ○ String variables
	• Usage of the data types
	• Storage size and value ranges of the data types

3.2 Select the appropriate data type for a given situation	• Determine appropriate data type based on user input or given value
3.3 Identify the correct syntax and usage for constants and variables (e.g., variable scope) in a program	• Variable scope • Naming conventions • Declaration and initialization of variables

3.4 Determine the correct syntax and safe functions for operations on strings, arrays, and data structures, including length, substring, and concatenation	• String operations
	  ○ Built-in functions
	  ▪ Length
	  ▪ Substring
	  ▪ Concatenation
	• Array operations
	  ○ Functions
	  ▪ Sort
	  ▪ Search
	  ○ Out-of-bounds exception errors

3.5 Explain complications of storing and manipulating data (i.e., the Big-O notation used to analyze storage and efficiency concerns, etc.)	● The Big-O notation used to analyze storage and efficiency concerns ● Memory usage ● Complexity algorithms (time to code) ● Efficient code (time to run)
3.6 Discuss data structure size concerns and memory management, including stack and heap	● Dynamic memory allocation ● Memory fragmentation/leaks ● Garbage collection
3.7 Implement file storage operations, including reading, writing, and creating files	● I/O File Operations ○ Open ○ Read ○ Write ○ Close ● File types
STANDARD 4.0 PERFORM BASIC COMPUTER MATHEMATICS IN INFORMATION TECHNOLOGY	
4.1 Apply basic mathematics to hardware and software design, logic, and variable scope (e.g., bits, bytes, kilobytes, megabytes, gigabytes, terabytes, and petabytes) including kilohertz, megahertz, and gigahertz	● Units of digital storage ○ Bits ○ Bytes ○ Kilobytes ○ Megabytes ○ Gigabytes ○ Terabytes ○ Petabytes ● Processors speeds ○ Kilohertz ○ Megahertz ○ Gigahertz
4.2 Calculate binary conversions (e.g., decimal, hexadecimal, and binary) to solve hardware and software problems	● Binary (Base 2) ● Decimal (Base 10) ● Hexadecimal (Base 16) ● Knowledge of various number systems ● Number system conversion
4.3 Identify and correctly use arithmetic operations applying the order of operations for programming [e.g., Parenthesis; Exponents;	● Apply arithmetic operators (+, -, *, /, %) ● Order of operations

Multiplication and Division, from left to right; and Addition and Subtraction, from left to right (PEMDAS)]	○ PEMDAS (parenthesis, exponents, multiplication, division, addition, subtraction) ○ BODMAS (brackets, orders (powers/exponents or roots), division, multiplication, addition, subtraction)
4.4 Interpret and construct mathematical formulas used in code [i.e., y = nx + b2 – n(a + b), etc.]	● Example: y = nx + b2 - n(a+b) ● Quadratic equation ● Problem understanding and equation formation
4.5 Identify correct and problematic uses of integers, floating-point numbers, and fixed-point numbers in arithmetic	● Various data types ○ Integers (whole numbers) ○ Float/double (decimals) ● Usage of the data types
4.6 Emphasize the importance of precision and accuracy in numerical computations to mitigate errors in software development	● Floating point errors ● Truncation/rounding errors
4.7 Investigate bit shift (left/right) and bitwise operations	● Divide or multiply an image by a power of two ● Graphic manipulation ● Computationally less expensive
STANDARD 5.0 UTILIZE CONDITIONAL STRUCTURES IN WRITING PROGRAMS	
5.1 Compare values using relational operators (e.g., =, >, <, >=, <=, and ≠)	● Relational operators ○ = ○ > ○ < ○ >= ○ <= ○ ≠ ● Various conditional operators and their results ● Difference between inclusive/exclusive limits (> and > =)
5.2 Evaluate Boolean expressions (e.g., AND, OR, NOT, NOR, and XOR)	● Boolean expressions ○ AND ○ OR ○ NOT ○ NOR ○ XOR ● Boolean expression evaluation (truth tables)

	• De Morgan's Law • Short-circuit (McCarthy) evaluation
5.3 Demonstrate and diagram conditional structures	• Flow charts • Pseudocode
5.4 Determine the correct syntax and nesting for decision structures (e.g., if/else, if, and switch case)	• If • If/else • Switch case • Controlling program flow based on various conditions • Nested if/if-else • Hierarchy of conditional check
5.5 Create and utilize functions and methods	• Function declaration • Use of parameters ○ Overloading/overriding • Return value • Void functions
STANDARD 6.0 UTILIZE BASIC DATA STRUCTURES AND ALGORITHMS IN WRITING PROGRAMS	
6.1 Demonstrate basic uses of arrays including initialization, storage, retrieval of values, and how to use them as arguments	• Array initialization and declaration • Types of arrays (i.e., integer, string, etc.) • Indices of array elements • How to use them as arguments
6.2 Distinguish between arrays and hash maps (associative arrays)	• Array values (index and values) vs. hash map (key and values) • Dictionaries • Ordered vs. unordered
6.3 Identify techniques for declaring, initializing, and modifying user-defined data types	• Declaring and initializing • Appending arrays and ArrayList • Class type ArrayList
6.4 Search and sort data in an array	• Search and sort algorithms (i.e., Bubble, Linear, Binary, Selection, etc.)

6.5 Diagram, create, and use two-dimensional arrays	● Initialize and declare 2D arrays ● Concept of rows and columns ● Format and access the elements of 2D array (a[][]) ○ Row-major and column-major ordering
6.6 Describe the efficiency of different sorting algorithms (e.g., bubble, insertion, and merge)	● Bubble ● Insertion ● Merge ● Selection ● Processing time/memory usage with different algorithms ● Best- and worst-case scenarios
6.7 Describe the efficiency of linear vs. binary searches [e.g., O(n) and O(log n)]	● O(n) ● O(log n)
6.8 Investigate more advanced data structures (i.e., trees, graphs, etc.)	● Trees ● Graphs ● Flow charts ● Purposes
STANDARD 7.0 UTILIZE ITERATIVE STRUCTURES IN WRITING PROGRAMS	
7.1 Identify various types of iteration structure (e.g., while, for, for-each, and recursion)	● Various types of iterations ○ While ○ For ○ For-each ○ Recursion ● Application for each type
7.2 Explain how loops are controlled (variable conditions and exits)	● Break/return keyword usage ● Control variable ● Increment/decrement control variable ● User input ● Count variable
7.3 Employ the correct syntax for nested loops	● Construct nested loop ● Control the inner loops through the outer loop
7.4 Compute the values of variables involved with nested loops (i.e.,	● Variable changes throughout a loop

variable changes throughout a loop, etc.)	• Code tracing ○ Working with the iteration variables and how they control each other's value (trace table)

7.5 Diagram iterative structures and use in writing programs	• Control structures
	  ○ Counting loop (For)
	  ○ Control variable (While)

STANDARD 17.0 EMPLOY OBJECT-ORIENTED PROGRAMMING TECHNIQUES	

17.1 Identify the differences between primitive and non-primitive data structures	• Primitive types
	  ○ Built-in data types, fixed sizes, stores actual values
	• Non-primitive types
	  ○ Reference data types, objects created dynamically

17.2 Differentiate between an object instance and a class	• Concept of Object-Oriented programming (OOP) • Class is the framework/blueprint • Object is an instance of the class with its own state and behavior ○ Object (i.e., Dog is an object of class Animal, etc.)
17.3 Discuss the roles of inheritance, composition, and class relationships	• A child class inherits or reuses instance variables and methods from a parent class • Is-a and has-a relationships
17.4 Instantiate objects from existing classes	• Syntax of object declaration and initialization • Using constructors with different signatures
17.5 Interpret the state of an object by invoking accessor methods	• Getter methods have a return value ○ Example: getStudentName() • A public method used to access a private instance variable
17.6 Change the state of an object by invoking a modifier method	• The setter method has no return value ○ Example: setStudentName("name") • A public method used to modify a private instance variable
17.7 Determine the requirements for constructing new objects by reading the documentation	• Create objects based on different constructor signatures

17.8 Create a user-defined class and a subclass of an existing class	• User-defined data type and methods • Subclass inherits methods and private instance variables from a superclass • Subclass specific data/methods only and reuse superclass methods • Subclass variables and methods override superclass variables and methods
17.9 Identify the use of an abstract class as opposed to an interface	• Interface vs. abstract class • Multiple inheritance • Default • Null
17.10 Explore advanced programming concepts (e.g., splitting files into different source files, principles of Inheritance, encapsulation, and polymorphism)	• Splitting files into different source files • Principles of Inheritance • Encapsulation • Polymorphism • Recursion
17.11 Investigate data representations in project creation (e.g., JSON and XML)	• JSON • XML
17.12 Explain the implementation and use of arguments, pointers, and references in programming	• Arguments • Pointers • References • Pass by value • Pass by reference
17.13 Demonstrate the principles of SOLID design and design patterns in object-oriented programming (OOP)	• SOLID design • Design patterns

Domain 2: Software/Application Development **Instructional Time: 30 - 40%**	
STANDARD 9.0 APPLY CLIENT-SIDE INTERNET SOFTWARE	
9.1 Examine key components and functions of the internet and web	• Internet

browsers	○ Clients and servers ○ Domain Name System (DNS) ○ IP Addressing ● Web browsers ○ History ○ Bookmarks ○ URL bar ● Basic and specialty browsers
9.2 Identify client collaboration sources/platforms (e.g., GitHub, Google Drive, Dropbox, JSFiddle, Visual Studio Live Share, and browser developer tools)	● Collaboration platforms ○ GitHub ○ Google Drive ○ Dropbox ○ JSFiddle ○ Visual Studio Live Share ● Browser developer tools ● Safety concerns
9.3 Analyze remote computing tools and services and their application [e.g., Secure Shell (SSH)]	● Secure Shell (SSH) ● Virtual Private Networks (VPNs) ● Remote desktop ● Remote application tools [i.e., OfficeSuite/Google Suite (i.e., docs, forms, sheets, etc.), Adobe Creative Suite, etc.]
9.4 Explore Modern Web and Application Frameworks (i.e., Node.js, Next.js, React, Django, Golang, Flutter, etc.)	● Node.js ● Next.js ● React ● Django ● Golang ● Flutter, etc.
9.5 Discuss containerization and microservices in the context of internet-based software architectures	● Comprehensive packages ● Modular components ● Development and deployment of internet-based software architectures
STANDARD 10.0 DEMONSTRATE PROGRAM ANALYSIS AND DESIGN	
10.1 Implement the steps in the System Development Life Cycle (SDLC) (e.g., planning, analysis, design, development, testing,	● System Development Life Cycle (SDLC) ○ Planning

implementation, and maintenance)	○ Analysis ○ Design ○ Development ○ Testing ○ Implementation ○ Maintenance
10.2 Develop program requirements/specifications and a testing plan (e.g., user stories, automated testing, and test procedures)	● User stories ● Automated testing ● Test procedures
10.3 Research industry-relevant programming languages (i.e., Java, JavaScript, Python, C#, etc.)	● Java ● JavaScript ● Python ● C#, etc.
10.4 Incorporate Rapid Application Development (RAD) and Rational Unified Process (RUP) methodologies	● Rapid Application Development (RAD) ● Rational Unified Process (RUP) methodologies
10.5 Investigate data handling, encryption requirements, and data requirements in program design by industry [i.e., HIPAA, Payment Card Industry Data Security Standard (PCI DSS), Gramm-Leach-Bliley Act (GLBA), etc.]	● HIPAA ● Payment Card Industry Data Security Standard (PCI DSS) ● Gramm-Leach-Bliley Act (GLBA) , etc.
10.6 Evaluate different database technologies (e.g., SQL vs. NoSQL and other emerging database standards)	● SQL vs. NoSQL ● Other emerging database standards
10.7 Explain the process of decomposing a large programming problem into smaller, more manageable procedures	● If statements with embedded conditions ● If/Else statements with embedded conditions
10.8 Demonstrate “visualizing” as a problem-solving technique (e.g. IDE tools) prior to writing code	● IDE tools ● Flowcharting
10.9 Describe problem-solving and troubleshooting strategies applicable to software development	● Common errors list ○ Variable names ○ Initializing error ○ Commenting out segments to identify section errors

10.10 Discuss the importance of using data and feedback to improve apps and decision-making	● Analysis of apps/programs ● Research add-ons for applications
STANDARD 11.0 DEVELOP A PROGRAM	
11.1 Discuss common editors and add-ins	● Parts of an Integrated Development Environment (IDE) ○ Source Code Editor ○ Compiler ○ Debugger ● Code completion ● Syntax highlighting
11.2 Use a program editor to enter and modify code	● Integrated Development Environment (IDE) ○ MIT App Inventor, Greenfoot, TextPad, BlueJ, Eclipse, etc.
11.3 Identify correct input/output statements	● Input ○ Save values into the appropriate variable type ● Output ○ Use programming language syntaxes ○ Case sensitive language errors
11.4 Choose the correct method of assigning input to variables including data sanitization (i.e., input text to numbers, etc.)	● Input text to numbers ○ Appropriate data types to variable assignment(s) ○ Conditional checks ○ Exception handling
11.5 Determine the correct method of outputting data with formatting and escape characters (e.g., ANSI escape code)	● ANSI escape code ● Language-specific output statements and escape characters
11.6 Differentiate between interpreted and compiled code and run executable code	● Interpreted code vs. compiled code ● Interpreted code/user language ● Compiled code/machine language ● Byte code/OOP languages
11.7 Identify the purpose of a build system (e.g., make, rake, ant, maven, SCons, and grunt)	● Compile and link source code into binary code with build tools ○ Make ○ Rake ○ Ant ○ Maven

	○ SCons ○ Grunt
11.8 Apply industry standards to program documentation (e.g., self-documenting code; function-level, program-level, and user-level documentation)	● Documentation levels ○ Self-documenting code ○ Function-level ○ Program-level ○ User-level documentation ● Document (with the code, within the code) ● Comments in code - third-party understanding
11.9 Name identifiers and formatting code by applying recognized conventions (e.g., camel casing)	● Naming conventions and practices in programming ○ Camel casing ● Indentations and whitespace to format code
11.10 Perform refactoring techniques to reduce repetitious code and improve maintainability	● Methods/functions to include blocks of code that can be reused ● Call/access these methods for code reusability
11.11 Use parameters to pass data into program modules and return values from modules	● Method calls with parameters ● Datatype of parameters ● The datatype of arguments passed into the module ● Use of return statements ○ The module processes data/parameters and can return data/output ○ How to catch the return values and use them further ○ The datatype of returning value to match exactly with the return data type in the module definition
11.12 Discuss the use of random number generators, including concepts of true randomness and seeding	● Random number generation ○ Starting value ○ Range of values ● Random generation vs. true randomness ○ Seed for random generation ○ Techniques used for true randomness
11.13 Apply pseudocode or graphical representations to plan the structure of a program or module [e.g., flowcharting, white boarding, and Unified Modeling Language (UML)]	● Pseudocode ● Flowcharting ● White boarding

	• Unified Modeling Language (UML)
11.14 Create and implement basic algorithms	• Algorithm design
STANDARD 12.0 TEST AND DEBUG TO VERIFY PROGRAM OPERATION	

12.1 Explain errors in program modules	• Rectify syntax errors while writing the program
	• IDEs that highlight/suggest these errors
	• Camel casing
	• Reading error statements to identify error type and location
	• Compile errors
	  ○ Programming language syntaxes
	  ○ Case-sensitive language errors
	• Runtime errors
	  ○ Logic error
	  ○ Input/output error
	  ○ Undefined object
	  ○ Division by zero
	  ○ Array Out of Bound Exception error

12.2 Identify boundary cases and generate appropriate test data	• Limits (upper and lower) for various data/variables • Array Out of Bound Exception error
12.3 Perform integration testing, including tests within a program, to protect execution from bad input or other run-time errors [e.g., continuous integration (CI) and automated testing]	• Continuous integration (CI) • Automated testing • Collaborative coding • Test blocks of code to avoid errors • Unit vs. integration testing strategies • Commenting out suspected problematic areas

12.4 Categorize, identify, and correct errors in code, including syntax, semantic, logic, and runtime	• Errors in code
	  ○ Syntax
	  ○ Semantic
	  ○ Logic
	  ○ Runtime
	• Techniques to debug errors
	  ○ Print statements
	  ○ Breakpoints
	  ○ Try catch blocks

12.5 Practice different methods of debugging (e.g., hand-trace code and real-time debugging tools)	● Hand-trace code ● Real-time debugging tools ● Debugging techniques (i.e., interactive, print, remote, etc.)
STANDARD 14.0 USE VERSION CONTROL SYSTEMS	
14.1 Compare version control system (e.g., Git and Mercurial)	● Version control systems: ○ Git ○ Mercurial ○ GitHub ○ Subversion (SVN) ○ Team Foundation Server (TFS)
14.2 Identify the purpose and types of version control systems (e.g., local, centralized, and distributed)	● Local ● Centralized ● Distributed ● Backup system for projects ● Collaboration-branching, merging ● Efficiency in contributions
14.3 Create new repositories and perform basic operations (e.g., adding, pushing, and pulling source code from repositories)	● Adding to repository ● Pushing source code from repositories ● Pulling source code from repositories ● Location option (i.e., local/central/cloud-based, etc.)
14.4 Explain version control branching and its uses	● Duplicate a program code for revision control purposes ● Parallel modifications between various team members ● Enhancing/experimenting with a program on a branch
14.5 Restore previous versions of code from the repository	● Pros/cons of backing up ● Ability to revert to prior version when errors occur
14.6 Research the principles of DevOps, DevSecOps, and Continuous Integration/Continuous Deployment (CI/CD) as part of version control and software development lifecycle	● Facilitate faster releases by using automation ● Limit the need for interventions ● Regular and implemented interventions
14.7 Integrate version control workflows (i.e., continuous deployment, continuous integration, etc.) using collaborative development practices	● Continuous deployment ● Continuous integration ● Working on own branch of a shared program

	• Working on own visual (sprite, background), audio, etc. of a shared program
14.8 Demonstrate document version control (i.e., commit messages, recovery from common errors, release notes, etc.)	• Commit messages • Push quantity and frequency • Recovery from common errors • Release notes as they relate to a project
STANDARD 15.0 APPLY USER DESIGN PRINCIPLES TO INCLUDE WEBSITES AND APPLICATIONS	

15.1 Investigate user-centered design (UCD), prototyping, and wireframing used during the design process	• Key Principles, Methods, and Techniques for User-Centered Design (UCD)
	  ○ Users are involved throughout the design and development process to ensure that the final product is usable, accessible, and enjoyable
	• Key Principles, Methods, and Techniques for Prototyping
	  ○ Create scaled-down versions or representations of a product before investing in full-scale development
	• Key Principles, Methods, and Techniques for Wireframing
	  ○ Create simplified visual representations of a product or interface to outline its structure, layout, and content without focusing on aesthetic details

15.2 Apply W3C standards and style conventions (e.g., HTML, CSS, and JavaScript)	• HTML(HyperText Markup Language) • CSS (Cascading Style Sheets) • JavaScript • W3C standards • Standards and style conventions /web development
15.3 Construct web pages and applications that are compliant with the Americans with Disabilities Act (ADA) and sections 504 and 508 standards (e.g., emphasize accessibility and inclusive design in user interface development)	• Emphasize accessibility and inclusive design in user interface development • Create and develop web pages using W3C conventions • ADA, Section 504 and 508 standards and requirements to web pages
15.4 Explain the concept of responsive design and applications (i.e., loading times, optimized images, etc.)	• Web page loading times • Optimized images • Optimal end-user experience on web page through responsive design

	• CSS to create a responsive design for various browsers/device compatibility • Industry application tools available (i.e., Bootstrap, Wirefly, etc.)
15.5 Employ graphics methods to create images at specified locations	• Method/functions to render/reposition an image • HTML Canvas
15.6 Choose correct graphical user interface (GUI) objects for input and output of data to the GUI interface (e.g., text boxes, labels, radio buttons, check boxes, dropdowns, and list boxes)	• Graphical User Interface (GUI) objects ○ Text boxes ○ Labels ○ Radio buttons ○ Checkboxes ○ Dropdowns ○ List boxes
15.7 Apply UI/UX design for multiple platforms including computers, mobile devices, and browsers	• Understand platform differences • Responsive design • Testing • Accessibility
15.8 Incorporate search engine optimization (SEO) and web optimization techniques	• Search Engine Optimization (SEO) friendly website structure • Keyword research and optimization • Quality content • Mobile optimization • Website monitoring and analytics
15.9 Integrate user testing and feedback loops to refine interface designs and improve user experience	• Identify goals, key performance indicators • Testing methods and scenarios • Analyze results and refine • Collect ongoing feedback
15.10 Analyze feature limitations and compatibility issues between different browsers and their versions	• Identify browsers • Test feature compatibility • Address issues, test across browsers • Follow web standards
15.11 Discuss framework and component libraries for cascading style sheets (CSS) and JavaScript	• Cascading style Sheets (CSS) ○ Bootstrap

	○ Foundation ● JavaScript ○ React ○ Angular ○ jQuery
STANDARD 18.0 EMPLOY RUNTIME AND ERROR HANDLING TECHNIQUES	
18.1 Investigate debugging techniques, error propagation and handling, and graceful degradation	● Print statements ● Unit testing ● Code reviews ● Exception handling ● Error logging ● Maintaining functionality through graceful degradation
18.2 Research causes for compilation and logical errors	● Stack overflow ● Forum post ● Online communities ● GitHub ● Developer documentation ● Blogs
18.3 Identify and resolve runtime errors	● Runtime errors vs. syntax errors ● Input validation ● Appropriate error message for unexpected returns values ● Alternative steps with try-catch-handle blocks ● Susceptible module within the try block
18.4 Describe error handling strategies based on severity	● Response and recovery procedure ● Cause/source of the runtime error from its name ● Source of the error ● Identify and fix strategies for the line of code/module responsible
18.5 Identify and resolve unexpected return values	● Employ conditional checks/boundaries ● Modify return variable type
18.6 Investigate standard exception classes and their uses	● Null pointer exception ● Arithmetic exception ● IO exception

	• Read detailed message
18.7 Demonstrate the application of exception classes (i.e., try, catch, throw, etc.)	• Try • Catch • Throw • Standard exceptions that occur at runtime • Standard exception handling classes • User-defined exception handling class • Custom exception handling program

Domain 3: Network/Security **Instructional Time: 5 - 10%**	
STANDARD 1.0 RECOGNIZE SECURITY ISSUES	
1.1 Identify common computer threats (e.g., viruses, phishing, suspicious email, social engineering, spoofing, identity theft, spamming, and AI	• Computer threats ○ Viruses/malware ○ Phishing ○ Suspicious email ○ Social engineering ○ Spoofing ○ Identity theft ○ Spamming ○ Artificial Intelligence (AI) ○ Distributed Denial of Service (DDoS) ○ SQL Injection (Structured Query Language) • Protection against the malware • Cyber safety
1.2 Describe potential vulnerabilities and risk management for information security [i.e., security information and event management (SIEM) software, OWASP's Top 10, common vulnerabilities and exposure (CVEs), etc.]	• Security Information and Event Management (SIEM) software • Open Web Application Security Project's (OWASP's) Top 10 • Common Vulnerabilities and Exposures (CVEs) ○ Scripting ○ Authentication ○ Injections
1.3 Identify procedures to maintain data integrity and security (e.g., lock the screen; report and delete unrecognized, suspicious emails; use	• Personal safety precautions/best practices ○ Lock the screen

trustworthy USB flash drives; and use approved software)	○ Report and delete unrecognized suspicious emails ○ Use trustworthy USB flash drives ○ Use approved software ○ Password-protected systems, applications, etc. ● End-user computer security risks ○ Public wi-fi risks ○ Weak passwords
1.4 Explain best practices to maintain integrity and security in software development [e.g., encryption, hashing, code signing, sandboxes, virtual machine (VM) containers, code versioning systems, and digital signatures]	● Encryption ● Hashing ● Code signing ● Sandboxes ● Virtual machine (VM) containers ● Code versioning systems ● Digital signatures ● Limited user access group
1.5 Describe methods for sanitizing user input to prevent issues (e.g., buffer overflows and SQL injection)	● Buffer overflows ● SQL injection (Structured Query Language)
1.6 Analyze the confidentiality, integrity, and availability (CIA) triad	● Model to guide policies of security information ● The role each element plays in the security plan development process
1.7 Explain how software defects relate to software security (e.g., buffer overflows and cross-site scripting)	● Buffer overflows ● Cross-site scripting ○ Untrusted scripts from trusted sources
STANDARD 8.0 IDENTIFY INTERNET PROTOCOLS AND OPERATIONS	
8.1 Explain the benefits of cloud-based computing	● Applications for use ● Cookies and cache in the settings ● Benefits of cloud-based computing ○ Scalability ○ Cost efficiency ○ Accessibility ○ Reliability ○ Security ○ Maintenance and updates ○ Environmental impact

8.2 Classify the components and functions of the common internet protocols (e.g., HTTP, HTTPS, SSH, SFTP, FTPS, IP addresses, IPV6, and IMAP)	● HTTP ● HTTPS ● SSH ● SFTP ● FTPS ● IP addresses ● IPV4 vs. IPV6 ● IMAP ● Basic internet terminology ● Open Systems Interconnection (OSI) model ● Layers of API and frameworks ● Secure vs. insecure browsing
8.3 Determine services run by web servers [e.g., scripting languages (client- and server-side scripting), serverless architectures, cloud computing, databases, and media]	● Scripting languages ○ Client- and server-side scripting ● Serverless architectures ● Cloud computing ● Databases ● Media ● Basic knowledge of CSS and JavaScript ● Front-end vs. back-end scripting
8.4 Identify performance issues (e.g., bandwidth, internet connection types, pages loading slowly, resolution, and size graphics)	● Bandwidth ● Internet connection types ● Pages loading slowly ● Resolution and size of graphics ● Latency vs. bandwidth with internet connectivity ● Embedded graphics issues (JPG, PNG, GIF) with connection/speed ● Browser options
8.5 Compare different cloud service models [Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Function as a Service (FaaS)]	● Software as a Service (SaaS) ● Platform as a Service (PaaS) ● Infrastructure as a Service (IaaS) ● Function as a Service (FaaS)
8.6 Identify Internet of Things (IoT) and common communication interfaces (e.g., Bluetooth, NFC, Wi-Fi, and LTE)	● Communication interfaces ○ Bluetooth ○ Near-field communication (NFC) ○ Wi-Fi

	○ Long Term Evolution (LTE)
STANDARD 16.0 STORAGE MANAGEMENT AND SECURITY	
16.1 Identify different data storage types (i.e., RAID, Cloud, SSD, HDD, Flash, tape, disk systems, etc.) and explain how they relate to designing and developing software applications	● RAID (Redundant Array of Inexpensive Disks) ● Cloud ● SSD (Solid State Drive) ● HDD (Hard Disk Drives) ● Flash ● Tape ● Disk systems, etc.
16.2 Discuss data backup and recovery, data integrity, and data privacy (e.g., blockchain, edge computing, and quantum storage)	● Blockchain ● Edge computing ● Quantum storage ● Version control systems
16.3 Read/write data from/to a sequential file or database [i.e., handling sequential files; database operations; create, read, update, delete (CRUD) operations, etc.]	● Handling sequential files ● Database operations ● Create, read, update, delete (CRUD) operations ● Database (DB) management and storage ● Basics of procedural languages (i.e., PL, SQL, etc.)
16.4 Differentiate among cloud storage, software storage, defined storage, file storage, block storage, object storage, memory, and cache storage in software applications	● Storage types and security concerns ● Financial concerns about storage and security
16.5 Demonstrate creating, reading, updating, and dropping a database	● Simple database ● Duplicate/incorrect entries ● Query run checks
16.6 Employ the proper use of database applications that work with different languages (e.g., MongoDB MQL, Microsoft Access SQL, and Oracle Databases SQL)	● MongoDB MQL ● Microsoft Access SQL ● Oracle Databases SQL

Domain 4: Business/Legal Issues **Instructional Time: 5 - 10%**	
STANDARD 2.0 EXAMINE LEGAL AND ETHICAL ISSUES RELATED TO INFORMATION TECHNOLOGY	
2.1 Explore intellectual property rights including software licensing and software duplication [e.g., Digital Millennium Copyright Act (DMCA), software licensing, and software duplication]	● Digital Millennium Copyright Act (DMCA) ● Software licensing ● Software duplication
2.2 Compare and contrast open source and proprietary systems in relation to legal and ethical issues (e.g., data pricing, use of public and private networks, social networking, industry-related data, and data piracy)	● Open source vs. proprietary systems ○ Data pricing ○ Use of public and private networks ○ Social networking ○ Industry-related data ○ Data piracy
2.3 Identify issues and regulations affecting computers, other devices, the internet, and information privacy [e.g., HIPAA, COPPA, CISPA, FERPA, PCI, GDPR, European Union's (EU) Digital Markets Act (DMA) and data brokers]	● HIPAA ● COPPA ● CISPA ● FERPA ● PCI ● GDPR ● European Union's (EU) Digital Markets Act (DMA) ● Data brokers
2.4 Explore ethical and responsible technology development including considerations of data privacy and tracking regulations (i.e., cookies, GDPR, AI, etc.)	● Cookies ● GDPR ● AI, etc.
STANDARD 13.0 UTILIZE AND CREATE COMMUNITY RESOURCES	
13.1 Integrate standard library functions	● Download and import standard libraries relevant to the programming language in use
13.2 Design code that incorporates third-party libraries (e.g., web-based and package managers)	● Web-based ● Package managers ● Various free/paid reusable components available for the said language (i.e., SDK in Java, etc.)

13.3 Explain and interact with an Application Program Interface (API)	• Types of APIs (Application Program Interface)
	  ○ Web APIs
	  ○ Operating System APIs
	  ○ Library APIs

13.4 Investigate using community information to solve problems (e.g., Stack Overflow, a public computer programming Q&A platform)	• Stack overflow • A public computer programming Q&A platform • Forum post • Online communities • Developer documentation
13.5 Create a README markdown file (.md) to document and explain basic install and usage steps	• GitHub • Project description and software instructions on basic installation and usage steps
STANDARD 19.0 EXPLORE BUSINESS ASPECTS IN SOFTWARE DEVELOPMENT	
19.1 Discuss software development methods (i.e., top-down, waterfall, Agile, etc.)	• Top-down • Waterfall • Agile • Iterative • Spiral
19.2 Explore the basics of app markets, including popular platforms (i.e., Google Play Store, Apple App Store, etc.)	• Google Play Store • Apple App Store, etc.
19.3 Identify components of a successful app	• Error-free/thoroughly tested • Clear design and usability interface • Meets/exceeds security benchmarks • Meets/exceeds support and updates
19.4 Discuss the cost of developing and launching an app	• SDLC design • Publishing costs • Time/cost of contributors
19.5 Research monetization strategies for apps (i.e., ads, in-app purchases, premium features, etc.)	• Ads • In-app purchases • Premium features, etc.

19.6 Investigate basic project management concepts in the software development process	● SDLC models ● Collaboration tools (GitHub projects)
19.7 Evaluate the importance of customer feedback in the software development cycle	● Research and Development team ● Forums and social networks
19.8 Research trends in the technology sector (i.e., blockchain, quantum computing, edge computing mobile apps, cloud computing, AI, etc.)	● Blockchain ● Quantum computing ● Edge computing mobile apps ● Cloud computing ● Artificial Intelligence (AI), etc.