Arizona CTE

Software and App Design 11.0202.00 (2024)

RECOGNIZE SECURITY ISSUES 1

1. Identify common computer threats (e.g., viruses, phishing, suspicious email, social engineering, spoofing, identity theft, spamming, and AI) 1.1
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.] 1.2
3. Identify procedures to maintain data integrity and security (e.g., lock the screen; report and delete unrecognized, suspicious emails; use trustworthy USB flash drives; and use approved software) 1.3
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] 1.4
5. Describe methods for sanitizing user input to prevent issues (e.g., buffer overflows and SQL injection) 1.5
6. Analyze the confidentiality, integrity, and availability (CIA) triad 1.6
7. Explain how software defects relate to software security (e.g., buffer overflows and cross-site scripting) 1.7

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] 2.1
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) 2.2
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] 2.3
4. Explore ethical and responsible technology development including considerations of data privacy and tracking regulations (i.e., cookies, GDPR, AI, etc.) 2.4

Skip to main content

UTILIZE PRIMITIVE DATA TYPES AND STRINGS IN WRITING PROGRAMS 3	1	**Differentiate among numeric, Boolean, character, string variables, and float and double** 3.1
	2	**Select the appropriate data type for a given situation** 3.2
	3	**Identify the correct syntax and usage for constants and variables (e.g., variable scope) in a program** 3.3
	4	**Determine the correct syntax and safe functions for operations on strings, arrays, and data structures, including length, substring, and concatenation** 3.4
	5	**Explain complications of storing and manipulating data (i.e., the Big-O notation used to analyze storage and efficiency concerns, etc.)** 3.5
	6	**Discuss data structure size concerns and memory management, including stack and heap** 3.6
	7	**Implement file storage operations, including reading, writing, and creating files** 3.7
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** 4.1
	2	**Calculate binary conversions (e.g., decimal, hexadecimal, and binary) to solve hardware and software problems** 4.2
	3	**Identify and correctly use arithmetic operations applying the order of operations for programming [e.g., Grouping symbols, Exponents/roots, Multiply/divide, Add/subtraction (GEMA) from left to right]** 4.3
	4	**Interpret and construct mathematical formulas used in code [i.e., y = nx + b2 – n(a + b), etc.]** 4.4
	5	**Identify correct and problematic uses of integers, floating-point numbers, and fixed-point numbers in arithmetic** 4.5
	6	**Emphasize the importance of precision and accuracy in numerical computations to mitigate errors in software development** 4.6
	7	**Investigate bit shift (left/right) and bitwise operations** 4.7
UTILIZE CONDITIONAL STRUCTURES IN WRITING PROGRAMS 5	1	**Compare values using relational operators (e.g., =, >, <, >=, <=, and ≠)** 5.1
	2	**Evaluate Boolean expressions (e.g., AND, OR, NOT, NOR, and XOR)** 5.2
	3	**Demonstrate and diagram conditional structures** 5.3

Skip to main content

	4 Determine the correct syntax and nesting for decision structures (e.g., if/else, if, and switch case) 5.4
	5 Create and utilize functions and methods 5.5
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 6.1
	2 Distinguish between arrays and hash maps (associative arrays) 6.2
	3 Identify techniques for declaring, initializing, and modifying user-defined data types 6.3
	4 Search and sort data in an array 6.4
	5 Diagram, create, and use two-dimensional arrays 6.5
	6 Describe the efficiency of different sorting algorithms (e.g., bubble, insertion, and merge) 6.6
	7 Describe the efficiency of linear vs. binary searches [e.g., O(n) and O(log n)] 6.7
	8 Investigate more advanced data structures (i.e., trees, graphs, etc.) 6.8
UTILIZE ITERATIVE STRUCTURES IN WRITING PROGRAMS 7	1 Identify various types of iteration structure (e.g., while, for, for-each, and recursion) 7.1
	2 Explain how loops are controlled (variable conditions and exits) 7.2
	3 Employ the correct syntax for nested loops 7.3
	4 Compute the values of variables involved with nested loops (i.e., variable changes throughout a loop, etc.) 7.4
	5 Diagram iterative structures and use in writing programs 7.5
IDENTIFY INTERNET PROTOCOLS AND OPERATIONS 8	1 Explain the benefits of cloud-based computing 8.1
	2 Classify the components and functions of the common internet protocols (e.g., HTTP, HTTPS, SSH, SFTP, FTPS, IP addresses, IPV6, and IMAP) 8.2
	3 Determine services run by web servers [e.g., scripting languages (client- and server-side scripting), serverless architectures, cloud computing, databases, and media] 8.3
	4 Identify performance issues (e.g., bandwidth, internet connection types, pages loading slowly, resolution, and size of graphics) 8.4

Skip to main content

	5 Compare different cloud service models [e.g., Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Function as a Service (FaaS)] 8.5
	6 Identify Internet of Things (IoT) and common communication interfaces (e.g., Bluetooth, NFC, Wi-Fi, and LTE) 8.6
APPLY CLIENT-SIDE INTERNET SOFTWARE 9	1 Examine key components and functions of the internet and web browsers 9.1
	2 Identify client collaboration sources/platforms (e.g., GitHub, Google Drive, Dropbox, JSFiddle, Visual Studio Live Share, and browser developer tools) 9.2
	3 Analyze remote computing tools and services and their application [e.g., Secure Shell (SSH)] 9.3
	4 Explore Modern Web and Application Frameworks (i.e., Node.js, Next.js, React, Django, Golang, Flutter, etc.) 9.4
	5 Discuss containerization and microservices in the context of internet-based software architectures 9.5
DEMONSTRATE PROGRAM ANALYSIS AND DESIGN 10	1 Implement the steps in the System Development Life Cycle (SDLC) (e.g., planning, analysis, design, development, testing, implementation, and maintenance) 10.1
	2 Develop program requirements/specifications and a testing plan (e.g., user stories, automated testing, and test procedures) 10.2
	3 Research industry relevant programming languages (i.e., Java, JavaScript, Python, C#, etc.) 10.3
	4 Incorporate Rapid Application Development (RAD) and Rational Unified Process (RUP) methodologies 10.4
	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.] 10.5
	6 Discuss different database technologies (e.g., SQL vs. NoSQL and other emerging database standards) 10.6
	7 Explain the process of decomposing a large programming problem into smaller, more manageable procedures 10.7
	8 Demonstrate “visualizing” as a problem-solving technique (e.g. IDE tools) prior to writing code 10.8
	9 Describe problem-solving and troubleshooting strategies applicable to software development 10.9

Skip to main content

	1 Discuss the importance of using data and feedback to improve apps and decision-making 10.1
DEVELOP A PROGRAM 11	1 Discuss common editors and add-ins 11.1
	2 Use a program editor to enter and modify code 11.2
	3 Identify correct input/output statements 11.3
	4 Choose the correct method of assigning input to variables including data sanitization (i.e., input text to numbers, etc.) 11.4
	5 Determine the correct method of outputting data with formatting and escape characters (e.g., ANSI escape code) 11.5
	6 Differentiate between interpreted and compiled code and run executable code 11.6
	7 Identify the purpose of a build system (e.g., make, rake, ant, maven, SCons, and grunt) 11.7
	8 Apply industry standards to program documentation (e.g., self-documenting code; function-level, program-level, and user-level documentation) 11.8
	9 Name identifiers and formatting code by applying recognized conventions (e.g., camel casing) 11.9
	1 Perform refactoring techniques to reduce repetitious code and improve maintainability 11.1
	11 Use parameters to pass data into program modules and return values from modules 11.11
	12 Use random number generators, including concepts of true randomness and seeding 11.12
	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)] 11.13
	14 Create and implement basic algorithms 11.14
TEST AND DEBUG TO VERIFY PROGRAM OPERATION 12	1 Explain errors in program modules 12.1
	2 Identify boundary cases and generate appropriate test data 12.2
	3 Perform integration testing, including tests within a program, to protect execution rom bad input or other run-time errors [e.g., continuous integration (CI) and utomated testing] 12.3

Skip to main content

	4 Categorize, identify, and correct errors in code, including syntax, semantic, logic, and runtime 12.4
	5 Practice different methods of debugging (e.g., hand-trace code and real-time debugging tools) 12.5
UTILIZE AND CREATE COMMUNITY RESOURCES 13	1 Integrate standard library functions 13.1
	2 Design code that incorporates third-party libraries (e.g., web-based and package managers) 13.2
	3 Explain and interact with an Application Program Interface (API) 13.3
	4 Investigate using community information to solve problems (e.g., Stack Overflow, a public computer programming Q&A platform) 13.4
	5 Create a README markdown file (.md) to document and explain basic install and usage steps 13.5
USE VERSION CONTROL SYSTEMS 14	1 Compare version control systems (i.e., Git, Subversion, etc.) 14.1
	2 Identify the purpose and types of version control systems (e.g., local, centralized, and distributed) 14.2
	3 Create new repositories and perform basic operations (e.g., adding, pushing, and pulling source code from repositories) 14.3
	4 Explain version control branching and its uses 14.4
	5 Restore previous versions of code from the repository 14.5
	6 Research the principles of DevOps, DevSecOps, and Continuous Integration/Continuous Deployment (CI/CD) as part of version control and the software development lifecycle 14.6
	7 Integrate version control workflows (i.e., continuous deployment, continuous integration, etc.) using collaborative development practices 14.7
	8 Demonstrate document version control (i.e., commit messages, recovery from common errors, release notes, etc.) 14.8
APPLY USER DESIGN PRINCIPLES TO INCLUDE WEBSITES AND APPLICATIONS 15	1 Investigate user-centered design (UCD), prototyping, and wireframing used during the design process 15.1
	2 Apply W3C standards and style conventions (e.g., HTML, CSS, and JavaScript) 15.2

Skip to main content

	3 Discuss 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) 15.3
	4 Explain the concept of responsive design and applications (i.e., loading times, optimized images, etc.) 15.4
	5 Employ graphics methods to create images at specified locations 15.5
	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) 15.6
	7 Apply UI/UX design for multiple platforms including computers, mobile devices, and browsers 15.7
	8 Incorporate search engine optimization (SEO) and web optimization techniques 15.8
	9 Integrate user testing and feedback loops to refine interface designs and improve user experience 15.9
	1 Analyze feature limitations and compatibility issues between different browsers and their versions 15.1
	11 Discuss framework and component libraries for cascading style sheets (CSS) and JavaScript 15.11
EXPLORE 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 16.1
	2 Discuss data backup and recovery, data integrity, and data privacy (e.g., blockchain, edge computing, and quantum storage) 16.2
	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.] 16.3
	4 Differentiate among cloud storage, software storage, defined storage, file storage, block storage, object storage, memory, and cache storage in software applications 16.4
	5 Demonstrate creating, reading, updating, and dropping a database 16.5
	6 Employ the proper use of database applications that work with different languages (i.e., MongoDB MQL, Microsoft Access SQL, Oracle Databases SQL, etc.) 16.6

Skip to main content

EMPLOY OBJECT-ORIENTED PROGRAMMING TECHNIQUES 17	**1** Identify the differences between primitive and non-primitive data structures 17.1
	2 Differentiate between an object instance and a class 17.2
	3 Discuss the roles of inheritance, composition, and class relationships 17.3
	4 Instantiate objects from existing classes 17.4
	5 Interpret the state of an object by invoking accessor methods 17.5
	6 Change the state of an object by invoking a modifier method 17.6
	7 Determine the requirements for constructing new objects by reading the documentation 17.7
	8 Create a user-defined class and a subclass of an existing class 17.8
	9 Identify the use of an abstract class as opposed to an interface 17.9
	1 Explore advanced programming concepts (e.g., splitting files into different source files, principles of Inheritance, encapsulation, and polymorphism) 17.1
	11 Investigate data representations in project creation (e.g., JSON and XML) 17.11
	12 Explain the implementation and use of arguments, pointers, and references in programming 17.12
	13 Demonstrate the principles of SOLID design and design patterns in object-oriented programming (OOP) 17.13
EMPLOY RUNTIME AND ERROR HANDLING TECHNIQUES 18	**1** Investigate debugging techniques, error propagation and handling, and graceful degradation 18.1
	2 Research causes for compilation and logical errors 18.2
	3 Identify and resolve runtime errors 18.3
	4 Describe error handling strategies based on severity 18.4
	5 Identify and resolve unexpected return values 18.5
	6 Investigate standard exception classes and their uses 18.6
	7 Demonstrate the application of exception classes (i.e., try, catch, throw, etc.) 18.7
EXPLORE BUSINESS ASPECTS IN SOFTWARE DEVELOPMENT 19	**1** Discuss software development methods (i.e., top-down, waterfall, Agile, etc.) 19.1
	2 Explore the basics of app markets, including popular platforms (i.e., Google Play Store, Apple App Store, etc.) 19.2

Skip to main content

3 **Identify components of a successful app** 19.3

4 **Discuss the cost in developing and launching an app** 19.4

5 **Research monetization strategies for apps (i.e., ads, in-app purchases, premium features, etc.)** 19.5

6 **Investigate basic project management concepts in the software development process** 19.6

7 **Evaluate the importance of customer feedback in the software development cycle** 19.7

8 **Research trends in the technology sector (i.e., blockchain, quantum computing, edge computing mobile apps, cloud computing, AI, etc.)** 19.8