**Software Engineering Glossary**

Welcome! This alphabetized glossary contains many of the terms you’ll find within this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are important for you to recognize when working in the industry, participating in user groups, and participating in other certificate programs.

| **Term** | **Definition** | **Video where the term is introduced** |
| --- | --- | --- |
| **2-tier** | 2-tier is a computing model in which the server hosts, delivers, and manages most of the resources and services delivered to the client. | Week 4: Architectural Patterns in Software |
| **3-tier** | The 3-tier pattern organizes applications into three logical and physical computing tiers: the presentation tier, or user interface, the middle tier which is usually the application tier where business logic is processed, and the data tier, where the data is stored and managed. | Week 4: Architectural Patterns in Software |
| **Agile** | Agile is an iterative method of software development. Teams work in cycles, or sprints and unit testing happens in each sprint. At the end of each sprint, a chunk of working code is released at a meeting where stakeholders can see the new functionality and provide feedback. | Week 1: Software Development Methodologies |
| **Alpha release** | The alpha release is the first functioning version of the system released to a select group of stakeholders. | Week 1: Building Quality Software |
| **API gateway** | The API Gateway routes the API from the client to a service. Orchestration handles communication between services. | Week 4: Architectural Patterns in Software |
| **Application deployment environment** | An application environment is the combination of the hardware and software resources required to run an application. | Week 4: Application Deployment Environments |
| **Application Programming Interface (API)** | An API is code that allows two programs to communicate. | Week 2: The importance of backend development |
| **Array** | In an array, a fixed number of elements of the same type are stored in sequential order, starting from index zero. | Week 3: Introduction to Programming Concepts Part 1 |
| **Assembly language** | Assembly language is a low-level programming language that uses a set of symbols to represent machine code. | Week 3: Query and Assembly Programming Languages |
| **Attributes** | An object’s properties and data are its attributes. | Week 4: Object-oriented Analysis and Design |
| **Backend developers** | Backend developers deal with everything that happens on the server before the code and data are sent to the client. | Week 2: Overview of Web and Cloud Development |
| **Behavioral models** | Behavioral models describe what a system does, without explaining how it does it. The overall behavior of a system can be communicated through behavioral models. | Week 4: Software Design and Modeling |
| **Beta release** | The beta release, also called a limited release, is given to the stakeholders outside of the developing organization. | Week 1: Building Quality Software |
| **Black-box testing** | Black box testing is a method of testing where the tester does not look at source code or internal structure. | Week 1: Software Testing |
| **Boolean expression** | A Boolean expression is a type of programming statement with only two values, either “true” or “false.” | Week 3: Branching and Looping Programming Logic |
| **Branching** | Branching logic is where a computer program makes a decision following a different set of instructions, depending on whether certain conditions are met during the program’s execution. | Week 3: Branching and Looping Programming Logic |
| **Build automation servers** | Build automation servers execute build automation utilities on a scheduled or triggered basis. | Week 2: More Application Development Tools |
| **Build automation utilities** | Build automation utilities generate build artifacts like executables, by compiling and linking source code. | Week 2: More Application Development Tools |
| **Business logic** | An application’s business logic dictates things like transaction results and what data is written to and retrieved from a database. | Week 4: Production Deployment Components |
| **Cascading Stylesheets (CSS)** | Developers use CSS to create stylish websites. CSS provides front-end developers with a standard method to define, apply, and manage different sets of style characteristics for a website and each of its components. CSS ensures uniformity in look and feel, style, colors, fonts, designs and layouts. | Week 2: Learning Front-end development |
| **Chief Technology Officer (CTO)** | The CTO oversees all of the research and development in a company. They monitor the company’s systems and infrastructure to ensure that they meet company needs and budget. | Week 5: Career Paths in Software Engineering |
| **Class** | A class is the generic version of an object. The class contains the object’s generic attributes – its properties and methods. | Week 4: Object-oriented Analysis and Design |
| **Class diagram** | Class diagrams are commonly used to communicate a software system’s structure in OOAD. The class diagram shows how the classes in an object-oriented design relate to one another. | Week 4: Object-oriented Analysis and Design |
| **Client-Server** | Another term for 2-tier. | Week 4: Architectural Patterns in Software |
| **Compiled language** | Compiled languages use a compiler program creates a program file, which runs the software. The compiler takes the source code and converts it from the programming language to machine code. Then it is compiled into one executable file. Finally, the program runs when you select the icon or file on your device. | Week 3: Interpreted and Compiled Programming Languages |
| **Component** | A component is an individual unit of encapsulated functionality that serves as a part of an application in conjunction with other components. | Week 4: Approaches to Application Architecture |
| **Component-based architecture** | Component-based architecture focuses on the decomposition of the design into logical components. Component-based architecture provides a higher level of abstraction than object-oriented designs. A component-based architecture should define, compose, and implement loosely coupled independent components so they work together to create an application. | Week 4: Approaches to Application Architecture |
| **Conditional statement** | An “if-then” statement that allows computers to use branching logic. | Week 3: Branching and Looping Programming Logic |
| **Constant** | A constant is a data item whose value does not change within a program. | Week 3: Introduction to Programming Concepts Part 1 |
| **Continuous Integration / Continuous Delivery or Continuous Integration / Continuous Deployment (CI/CD)** | CI/CD refers to the practices of continuous integration and either continuous delivery or continuous deployment. CI/CD is a best practice for DevOps teams enabling developers to deliver frequent changes reliably. Continuous Integration (CI) ensures that all the code components work together smoothly. Continuous delivery (CD) begins where CI ends. The CI process automatically builds and tests your code, then CD deploys all code changes in a build to a testing or staging environment. | Week 2: More Application Development Tools |
| **Create, Read, Update, Delete (CRUD)** | CRUD is shorthand referring to the most common types of queries: create, read, update, and delete. | Week 3: Query and Assembly Programming Languages |
| **Database Management System (DBMS)** | The DBMS controls a database by connecting the database to users or other programs. | Week 4: Production Deployment Components |
| **Distributed system** | A distributed system is a system with multiple services located on different machines that coordinate interactions by passing messages to each other via a communication protocol such as hypertext transfer protocol, also known as HTTP. | Week 4: Approaches to Application Architecture |
| **Driver/navigator** | Driver/navigator is a pair programming style where one developer is the driver, typing in the code, and the other is the navigator, reviewing the code as it’s written and giving directions on where to go next. | Week 2: Pair Programming |
| **Dynamic content** | Dynamic content is generated each time it is requested by the client. | Week 2: Overview of Web and Cloud Development |
| **Encapsulation** | Encapsulation consists of bundling a component’s data and methods to hide its internal state, so it doesn’t expose its specific implementation. | Week 4: Approaches to Application Architecture |
| **Event-driven** | The event-driven pattern focuses on producers and consumers of events. Producers listen for and react to triggers while consumers process an event. The producer publishes the event to an event router. The router determines which consumer to push the event to. The triggering event generates a message, called an event notification, to the consumer which is listening for the event. | Week 4: Architectural Patterns in Software |
| **Extensibility** | Extensibility entails the ability to add behavior to a component without changing other components. | Week 4: Approaches to Application Architecture |
| **Firewall** | Firewalls prevent unauthorized access to or from a private network. | Week 4: Application Deployment Environments |
| **Framework** | Frameworks provide a standard way to build and deploy applications. You can think of a framework as being a skeleton that you can extend by adding your own code, providing a scaffold on which to build your apps. | Week 2: Introducing Application Development Tools |
| **Frontend developers** | Frontend developers deal with everything that happens on the client side. It is everything the user can see and interact with. | Week 2: Overview of Web and Cloud Development |
| **Fullstack developers** | Fullstack developers have skills, knowledge, and experience in both frontend and backend environments. | Week 2: Overview of Web and Cloud Development |
| **Function** | A function is a piece of structured, stand-alone, and reusable code that will perform a single specific action. Functions take in data as an input, then process the data, and then return the result as an output. | Week 3: Introduction to Programming Concepts Part 2 |
| **Functional testing** | Functional testing is a type of black box testing which is concerned with testing inputs and outputs. | Week 1: Software Testing |
| **General availability** | A stable version of the software intended for all users. | Week 1: Building Quality Software |
| **Git** | Git is a code repository where you can track changes, split code into different branches for more focused development, and then merge them back into the main body of code. | Week 2: Introducing Application Development Tools |
| **Hybrid Cloud** | A mix of both public and private clouds, working together seamlessly, is called a hybrid cloud model. | Week 4: Application Deployment Environments |
| **Hypertext Markup Language (HTML)** | HTML is used to create the physical structure of a website. The physical structure contains elements such as text, links, images/videos, page dividers and buttons. | Week 2: Learning Front-end development |
| **Information developer** | Information developer is another term for technical writer. | Week 1: Roles in Software Engineering Projects |
| **Inheritance** | A subclass inherits the same properties and methods as its parent class but also may add its own additional properties and methods. | Week 4: Object-oriented Analysis and Design |
| **Integrated Development Environment (IDE)** | IDEs make it easier to build and manage code. Good IDEs support multiple languages and integrate with management and storage tools like Git and GitHub. Other useful IDE features include custom extensions and themes for supporting your working style and environment. | Week 2: Overview of Web and Cloud Development |
| **Integration testing** | Integration testing is a testing stage that occurs after unit testing, when the components are integrated into a larger product. | Week 1: Building Quality Software |
| **Interpreted language** | Interpreted languages are commonly referred to as scripted or scripting languages. Programs written in interpreted or scripted language, like Python and HTML, run through the programming interpreter on your computer’s operating system or in your web browser. The interpreter takes the human-readable scripted code and then translates it into machine code, enabling the computer to complete the requested task. | Week 3: Interpreted and Compiled Programming Languages |
| **JavaScript** | JavaScript is an object-oriented programming language that is used in conjunction with HTML and CSS to add interactivity to a website. | Week 2: Learning Front-end development |
| **Learner Stylesheets (LESS)** | LESS enhances CSS, adding more styles and functions. | Week 2: Learning Front-end development |
| **Library** | Libraries are collections of code, like standard programs and subroutines, that you can use within your code. | Week 2: Introducing Application Development Tools |
| **Load** | Load refers to the number of concurrent users, the number of transactions, and the amount of data transferred back and forth between the clients and servers. | Week 4: Application Deployment Environments |
| **Load balancer** | Load balancers distribute network traffic efficiently amongst multiple servers on a network. Load balancers are used to prevent server traffic overload and are located between clients and the servers. A load balancer determines which servers are capable of fulfilling those requirements in a manner that maximizes availability and responsiveness. | Week 4: Production Deployment Components |
| **Loop** | A loop is a sequence of instructions that continually repeats until reaching a specific condition. | Week 3: Branching and Looping Programming Logic |
| **Method** | Another term for function. | Week 3: Introduction to Programming Concepts Part 2 |
| **Microservices** | Microservices are an approach to building an application that breaks its functionality into modular components called services. | Week 4: Architectural Patterns in Software |
| **Minimal Viable Product (MVP)** | The MVP is a basic feature set software release given to stakeholders so they can provide feedback on the basic feature set. The MVP contains a feature set to validate assumptions about the software. | Week 1: Software Development Methodologies |
| **Node** | A node is any device on a network that can recognize, process, and transmit data to other nodes on the network. | Week 4: Approaches to Application Architecture |
| **Non-functional testing** | Non-functional testing tests an application for attributes like performance, security, scalability, and availability. Non-functional testing checks to see if the SUTs non-functional behavior is performing properly. | Week 1: Software Testing |
| **Object-oriented Analysis and Design (OOAD)** | OOAD is is an approach for analyzing and  designing a software system when the system will use object-oriented programming languages to develop it. | Week 4: Object-oriented Analysis and Design |
| **Object-Oriented Programming (OOP)** | A programming methodology that is focused on objects rather than functions, which is what procedure-oriented programming is focused on. The objects themselves will contain data in the form of properties (or attributes) and code in the form of procedures (or methods). The key distinction between the two methodologies is that where procedural programming uses methods to operate on separate data structures, OOP packages them both together, so an object operates on its own data structure. | Week 3: Introduction to Programming Concepts Part |
| **Object-Relational Mapping (ORM)** | Tools used to connect the database and retrieve the correct data. | Week 2: The importance of backend development |
| **On-premises deployment** | For on-premises software deployments, an organization is responsible for the system, hardware, related infrastructure, and maintenance required to run the application. | Week 4: Application Deployment Environments |
| **Ops engineer** | Ops engineer is another term for site reliability engineer. | Week 1: Roles in Software Engineering Projects |
| **Package manager** | Package managers take care of the tasks of finding, installing, maintaining, or uninstalling software packages at the user’s request. | Week 2: More Application Development Tools |
| **Packages** | Packages are archive files that contain the app files, instructions for installation, and any metadata that you choose. They have their own metadata, including the package description, package version, and any dependencies, like other packages that need to be installed beforehand. | Week 2: More Application Development Tools |
| **Pair programming** | Pair programming is an extension of teamwork where two developers work side-by-side at one computer. | Week 2: Pair Programming |
| **Parameter** | The data that is passed into a function. | Week 3: Introduction to Programming Concepts Part 2 |
| **Peer-to-peer (P2P)** | The peer-to-peer architecture consists of a decentralized network of nodes that are both clients and servers. The workload is partitioned among these nodes. | Week 4: Architectural Patterns in Software |
| **Ping-pong** | Ping-pong is a pair programming style that incorporates test-driven development. For each task, one developer writes a failing test and then the second developer writes code to pass that test. The two developers work together at the end of each task refactoring the successful code to refine and improve it. | Week 2: Pair Programming |
| **Principal Engineer** | A principal engineer is responsible for the architecture and design of a software solution, as well as the development of it. They are expected to create processes and procedures for your team and provide technical direction. | Week 5: Career Paths in Software Engineering |
| **Private Cloud** | With a private cloud, the cloud infrastructure is provisioned for exclusive use by a single organization. The software system can be run on-premises, or the infrastructure could be owned, managed, and operated by a service provider. | Week 4: Application Deployment Environments |
| **Procedure** | Another term for function. | Week 3: Introduction to Programming Concepts Part 2 |
| **Product manager** | Product manager is similar to a product owner. The term product manager is typically used in a waterfall or waterfall-like software development project. | Week 1: Roles in Software Engineering Projects |
| **Product owner** | The product owner has the vision of what the product should look like. They have an intimate understanding of the client’s requirements, and the end-user’s needs. They are responsible for leading development efforts to create the software and for ensuring the product provides the value stakeholders are looking for. | Week 1: Roles in Software Engineering Projects |
| **Production environment** | The production environment is comprised of the infrastructure that runs and delivers the application to the end-user such as the servers, load balancers, and databases. | Week 4: Introduction to Software Architecture |
| **Production environment** | The production environment includes the entire solution stack consisting of both hardware and software on which the application runs as additional infrastructure components. The production environment is intended for all users. | Week 4: Application Deployment Environments |
| **Project manager** | A project manager makes sure a project runs smoothly and facilitates communication about the project. The project manager often deals with big picture issues such as planning, scheduling, and budgeting, allocating personnel and resources, executing the software plan, and team communication. | Week 1: Roles in Software Engineering Projects |
| **Prototype** | A prototype is a small-scale replica of the end product used to get stakeholder feedback and establish requirements. | Week 1: Phases of the SDLC |
| **Proxy server** | A proxy server is an intermediate server that sits in between two tiers and handles requests between those tiers. A proxy server can serve multiple purposes such as load balancing, system optimization, caching, acting as a firewall, obscuring the source of the request, encryption, scanning for malware, and more. | Week 4: Production Deployment Components |
| **Pseudocode** | Pseudocode provides a beneficial bridge to the project code because it closely follows the logic that the code will. | Week 3: Understanding Code Organization Methods |
| **Public Cloud** | The public cloud is when you leverage the software’s supporting infrastructure over the open internet on hardware owned by the cloud provider. | Week 4: Application Deployment Environments |
| **QA engineer** | QA engineers are in-charge of ensuring the quality of the product and that the software solution meets customer requirements. They are responsible for writing and executing test cases to identify bugs or deficiencies and provide this feedback to the development teams. | Week 1: Roles in Software Engineering Projects |
| **Query** | A query uses predefined and understandable instructions to make a request to a database. | Week 3: Query and Assembly Programming Languages |
| **Prototype** | A prototype is used to test basic design ideas. | Week 1: Software Testing |
| **Regression testing** | Regression testing, also called maintenance testing, confirms that a recent change to the application, such as a bug fix, does not adversely affect already existing functionality. | Week 1: Software Testing |
| **Responsive design** | Responsive design of a website means that it will automatically resize to the device it is being accessed from. | Week 2: Learning Front-end development |
| **Scalability** | Scalability is the application’s ability to dynamically handle the load as it or shrinks without it affecting the application’s performance. | Week 4: Application Deployment Environments |
| **Scripting language** | Another term for an interpreted language. | Week 3: Query and Assembly Programming Languages |
| **Scrum master** | The scrum master is focused on ensuring team and individual success. | Week 1: Roles in Software Engineering Projects |
| **Service** | A service is like a component, also a unit of functionality, but it is designed to be deployed independently and reused by multiple systems. A service focuses on a solution to a business need. | Week 4: Approaches to Application Architecture |
| **Service-oriented architecture (SOA)** | In a service-oriented architecture, or SOA, services are loosely coupled and interface with each other via a communication protocol over a network. SOA supports building distributed systems that deliver services to other applications through the communication protocol. | Week 4: Approaches to Application Architecture |
| **Site reliability engineer** | A site reliability engineer, sometimes called an SRE or ops engineer, bridges development and operation by combining software engineering expertise with IT systems management. They track incidents and facilitate meetings to discuss them; They also automate systems, procedures, and processes; assist with trouble shooting; and ensure reliability for the customer. | Week 1: Roles in Software Engineering Projects |
| **Soft skills** | Soft skills are personal characteristics and interpersonal skills. They’re the non-technical skills are harder to define, quantify, or certify than hard skills. | Week 5: Skills Required for Software Engineering |
| **Software architecture** | Software architecture serves as a blueprint for the software system that the programmers use to develop the interacting components of the software. The architecture comprises the fundamental structures of a software system and explains the behavior of that system. The architecture defines how components should interact with each other, the operating environment, and the principles used to design the software. | Week 4: Introduction to Software Architecture |
| **Software design document (SDD)** | The SDD is a collection of technical specifications that indicate how the design should be implemented. It provides a functional description of the software and design considerations such as assumptions, dependencies, constraints, requirements, objectives, and methodologies. | Week 4: Introduction to Software Architecture |
| **Software Development Lifecycle (SDLC)** | The SDLC is a systematic process to develop high-quality software in a predictable timeframe and budget. The goal of the SDLC is to produce software that meets a client’s business requirements. | Week 1: Introduction to the SDLC |
| **Software Requirement Specification (SRS)** | A document that combines and lists stakeholders’ requirements for an application. | Week 1: Phases of the SDLC |
| **Software stack** | A software stack is a combination of technologies that includes software and programming languages. The set of individual technologies is stacked in a hierarchy and works together to support the execution of an application. The higher levels in the stack provide tasks or services for the user and the lower levels interact with the computer hardware. Software stacks typically include Frontend technologies such as programming languages, frameworks, and user interface tools. And backend technologies such as programming languages, frameworks, web servers, app servers, operating systems, messaging applications, and databases. | Week 2: Introduction to Software Stacks |
| **Squad** | A squad is a small team of up to 10 developers. It is likely to consist of: A squad leader who acts as the anchor developer and coach for the squad. And a few software engineers who develop and implement the product features and test cases. It may also include one or two user experience developers or designers. | Week 2: Teamwork and Squads |
| **Staging environment** | The staging environment is the environment that is as close to replicating the production environment as possible but is not meant for general users. | Week 4: Application Deployment Environments |
| **Stakeholder** | The stakeholder is mainly responsible for defining project requirements and providing feedback if team members need clarification on requirements or if a proposed solution cannot be solved as planned. | Week 1: Roles in Software Engineering Projects |
| **Standard Operating Procedure (SOP)** | The SOP is written documentation that explains step-by-step how to accomplish a common, yet complex task that is organization specific. | Week 1: Software Documentation |
| **Static content** | Static content is content stored on a server. | Week 2: Overview of Web and Cloud Development |
| **Strong style** | Strong style is a pair programming style junior software engineers to learn from more experienced ones. So, the more experienced of the pair is the navigator and the driver learns from witnessing their implementation and thought processes. | Week 2: Pair Programming |
| **Structured design** | Structured design conceptualizes a software problem into well-organized smaller solution elements called modules and sub-modules. Structured design stresses organization in order to achieve a solution. | Week 4: Software Design and Modeling |
| **Structured Query Language (SQL)** | The most popular language used to query databases. | Week 3: Query and Assembly Programming Languages |
| **Syntactically Awesome Stylesheets (SASS)** | SASS enables you to use things like variables, nested rules, and inline imports to keep things organized. SASS allows you to create style sheets faster and more easily than CSS. | Week 2: Learning Front-end development |
| **System architect** | The system architect, designs and describes the architecture of a project as well as communicates that architecture to team members. They are responsible for designing the essential characteristics of the inner structure and technical aspects of the software. | Week 1: Roles in Software Engineering Projects |
| **System Requirement Specification (SysRS)** | The SysRS contains system capabilities, interfaces, and user characteristics in addition to what the SRS contains. It may include policy requirements, regulation requirements, personnel requirements, performance requirements, security requirements, and system acceptance criteria. It also outlines expectations of the hardware needed for the system in addition to software requirements. | Week 1: Requirements |
| **System testing** | System testing is a testing stage that occurs when the application is deemed completed. | Week 1: Building Quality Software |
| **System Under Test (SUT)** | The SUT is the software application when it is being tested. | Week 1: Software Testing |
| **Technical Architect** | Another term for a principal engineer. | Week 5: Career Paths in Software Engineering |
| **Technical lead** | Technical leads manage a team of developers and engineers developing the software in an organization. They are responsible for the entire development lifecycle and report to the project stakeholders. | Week 5: Career Paths in Software Engineering |
| **Technical writer** | The technical writer writes documentation for the end-user. | Week 1: Roles in Software Engineering Projects |
| **Test case** | Test cases are written to verify the functionality of a software application and ensure requirements have been satisfied. A test case contains steps, inputs, data, and the expected corresponding outputs. | Week 1: Software Testing |
| **Unified Modeling Language (UML) diagrams** | UML diagrams are diagrams that communicate structure and behavior using common programming language agnostic notation. UML diagrams will also be discussed in more detail in another video. | Week 4: Introduction to Software Architecture |
| **Unit testing** | Unit testing is often done by the developer and tests the smallest component of code that can be isolated from the rest of the system. | Week 1: Building Quality Software |
| **Use cases** | A use case is a description of how a user may interact with the software system. A use case tends to be more granular than a user story and describes how the system acts rather than the goal of the user’s action. | Week 1: Requirements |
| **User Acceptance Testing (UAT)** | UAT is a testing stage where the software is tested by the intended user. | Week 1: Building Quality Software |
| **User experience (UX) designer** | The UX designer balances making software intuitive but also as robust as it needs to be to address requirements. They define how the software behaves from the user’s perspective. | Week 1: Roles in Software Engineering Projects |
| **User Requirement Specification (URS)** | The URS describes the business needs and expectations of the end-users of the software system. | Week 1: Requirements |
| **User stories** | A user story is an explanation of a software requirement written from the perspective of the end-user or customer. A user story is stated in terms of the goal of the user’s actions. | Week 1: Requirements |
| **Variable** | Variables have assigned values that are passed into a function or subroutine within a more extensive program. A variable has a value that can change, depending on conditions or information passed to the program. | Week 3: Introduction to Programming Concepts Part 1 |
| **Vector** | Similar to arrays but vectors have a dynamic size, and they will automatically resize themselves as you add elements to them or remove elements from them. | Week 3: Introduction to Programming Concepts Part 1 |
| **V-shape model** | The V-shape model is a method of software development that contains “validation” phases and “verification” phases. The V-shape model is like waterfall in that it is also sequential. Each phase in verification corresponds with a validation phase. | Week 1: Software Development Methodologies |
| **Waterfall** | Waterfall is a sequential method of software development where the output of one phase is the input for the next phase of the cycle. | Week 1: Software Development Methodologies |

| **Term** | **Definition** |
| --- | --- |
| **AI** | Artificial intelligence |
| **Blockchain** | An immutable network allowing members to view only those transactions that are relevant to them |
| **Broad Network Access** | Cloud computing resources can be accessed through the network |
| **Cloud computing** | A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction |
| **GCP** | Google Cloud Platform |
| **Hypervisor** | A small software layer that enables multiple operating systems to run alongside each other, sharing the same physical computing resources |
| **IDC** | International Data Corporation |
| **IoT** | Internet of things |
| **Measured Service** | You only pay for what you use or reserve as you go |
| **NIST** | National Institute for Standards and Technology |
| **PaaS** | Platform as a service |
| **Pay-As-You-Go** | Users can order cloud resources from a larger pool of available resources and pay for them on a per-use basis |
| **POP** | Post office protocol |
| **Rapid elasticity** | You can increase or decrease resources as per your demand because of the elastic property of the cloud |
| **SaaS** | Software as a service |
| **Utility model of billing** | You are charged after the usage and at the end of the pre-defined period |
| **VM** | Virtual machine |

| **Term** | **Definition** |
| --- | --- |
| **BPM** | Business Process Management |
| **Composite multicloud** | A variant of hybrid multicloud, distributes single applications across multiple providers, allowing you to move application components across cloud services and vendors as needed |
| **CRM** | Customer Relationship Management |
| **HCM** | Human Capital Management |
| **Hybrid cloud** | A computing environment that connects an organization’s on-premises private cloud and third-party public cloud into a single, flexible infrastructure for running the organization’s applications and workloads |
| **Hybrid monocloud** | A hybrid cloud with one cloud provider |
| **Hybrid multicloud** | An open standards-based stack that can be deployed on any public cloud infrastructure |
| **IaaS** | Infrastructure as a service is a form of cloud computing that delivers fundamental compute, network, and storage resources to consumers on-demand, over the network, on a pay-as-you-go basis |
| **IoT** | Internet of things |
| **MDM** | Master Data Management |
| **PaaS** | Platform as a service is a cloud computing model that provides customers a complete platform—hardware, software, and infrastructure—to develop, deploy, manage, and run applications created by them or acquired from a third-party |
| **Pay-as-you-go** | Users can order cloud resources from a larger pool of available resources and pay for them on a per-use basis |
| **Private Cloud** | Cloud infrastructure provisioned for exclusive use by a single organization comprising multiple consumers, such as the business units within the organization |
| **Public cloud** | Users get access to servers, storage, network, security, and applications as services delivered by cloud service providers over the internet |
| **SaaS** | Software as a service is a cloud offering that provides users with access to a service provider’s cloud-based software |
| **SIP** | SaaS integration platforms |
| **TCO** | Total cost for ownership |
| **VM** | Virtual machine |
| **VPC** | Virtual Private Cloud |

| **Term** | **Definition** |
| --- | --- |
| **ACL** | Access Control Lists |
| **AZ** | Availability Zones are distinct Data Centers with their own power, cooling and networking resources. These Zones can have names like DAL-09 or us-east-1. |
| **Bare-metal hypervisor** | Installed directly on top of the physical server. They're more secure, have lower latency, and are usually the ones you see in the market the most |
| **Block storage** | Is presented to compute nodes using high-speed fiber connections, which means that read and write speeds are typically much faster and reliable than with file storage |
| **CDNs** | Content Delivery Networks is a distributed server network that delivers temporarily stored, or cached, copies of website content to users based on the user’s geographic location |
| **Cloud Region** | A geographic area or location where a Cloud provider’s infrastructure is clustered, and may have names like NA South or US East |
| **Containers** | Are an executable unit of software in which application code is packaged, along with its libraries and dependencies, in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud |
| **Data center** | A huge room or a warehouse containing cloud infrastructure |
| **Dedicated hosts** | Offer single-tenant isolation |
| **Direct Attached storage** | Or Local storage is storage which is presented directly to a cloud-based server and is effectively either within the host server chassis or within the same rack |
| **File storage** | Is typically presented to compute nodes as ‘NFS Storage’. NFS stands for Network File System and means that the storage is connected to compute nodes over a standard ethernet network |
| **Hosted hypervisor** | There's a layer of host OS between the physical server and the hypervisor. These hypervisors are less frequently used and mostly used for “end-user” virtualization |
| **HPC** | High-performance computing |
| **Hypervisor** | A piece of software that runs above the physical server or host |
| **IOPS** | Input/Output Operations Per Second and refers to the speed at which the disks can write and read data |
| **NFS** | Network File Storage |
| **Object storage** | Storage not attached to a compute node, rather it is accessed via an API |
| **Reserved virtual server** | Instances allow you to reserve capacity and guarantee resources for future deployments |
| **SDN** | Software Defined Networking |
| **Shared or Public Cloud VMs** | Are provider-managed, multi-tenant deployments that can be provisioned on-demand with predefined sizes |
| **Transient or Spot VMs** | Take advantage of unused capacity in a cloud data center |
| **Virtualization** | Process of creating a software-based or virtual version of something whether that be compute, storage, networking, servers, or applications |
| **VLANs** | Virtual Local Area Networks |
| **VM** | Virtual machines are software-based computers, based on virtualization technologies |
| **VPC** | Virtual Private Cloud |
| **VPN** | Virtual Private Networks |

| **Term** | **Definition** |
| --- | --- |
| **API** | Application Programming Interface |
| **Application modernization** | Helps organizations accelerate their digital transformation, take advantage of new technologies and services, and become more responsive to changing market dynamics |
| **Cloud native application** | An application developed from the outset to work only in the cloud environment, or an existing app that has been refactored and reconfigured with cloud native principles |
| **Continuous delivery** | Delivering small, well-designed, high-quality increments of software to customers |
| **Continuous deployment** | Progressing each new packaged build through the deployment lifecycle as rapidly as possible |
| **Continuous integration** | Creating packaged builds of the code changes released as immutable images |
| **Continuous monitoring** | Monitoring with tools that help developers understand the performance and availability of their applications, even before they’re deployed to production |
| **Delivery pipeline** | An automated sequence of steps that involves the stages of Ideation, Coding, Building, Deploying, Managing, and Continuous Improvement |
| **DevOps** | Collaborative approach where business owners and the development, operations, and quality assurance teams collaborate to continuously deliver software |
| **Hybrid multicloud** | An open standards-based stack that can be deployed on any public cloud infrastructure |
| **Microservices** | Break down large applications into their core functions |
| **Microservices architecture** | Approach in which a single application is composed of many loosely coupled and independently deployable, smaller components or services |
| **Monolithic architecture** | Approach in which a single application is built out of one piece of software |
| **Serverless** | Approach to computing that offloads responsibility for common infrastructure management tasks |
| **Service discovery** | Creates a roadmap for microservices to communicate |

| **Term** | **Definition** |
| --- | --- |
| **Access group** | A group of users and service IDs is created so that the same access can be assigned to all entities within the group with one or more access policies |
| **Administrative users** | Create, update, and delete application and service instances, and need insight into their team members’ activities |
| **API keys** | Unique identifiers are passed into an API to identify calling application or user |
| **Application Performance Monitoring** **(APM)** | Measures application availability and performance, providing tools needed to troubleshoot issues in an application's environment |
| **Application users** | Users of the cloud-hosted applications |
| **AppSec** | Application Security |
| **Audit and compliance** | A critical service within identity and access framework used to validate implemented controls against policies |
| **Authentication** | Also known as identity service, it enables applications deployed to the cloud to authenticate users at an application level |
| **BYOK** | Bring Your Own Keys |
| **Client-side encryption** | Occurs before data is sent to cloud storage |
| **Cloud directory services** | Used to securely manage user profiles and associated credentials inside a cloud environment |
| **Cloud encryption** | Also known as the last line of defense, it encrypts data and provides robust data access control, key management, and certificate management |
| **Cloud monitoring solutions** | Assess data, application, and infrastructure behaviors for performance, resource allocation, network availability, compliance, and security risks and threats |
| **Cloud security** | Policies, technological procedures, services, and solutions designed to secure the enterprise applications and data on the cloud against insider threats, data breaches, compliance issues, and organized security threats |
| **Database monitoring tools** | Help track processes, queries, and availability of services to ensure the accuracy and reliability of database management systems |
| **Decryption key** | Defines how the encrypted data will be transformed back to legible data |
| **Developer users** | Authorized to read sensitive information and to create, update, and delete applications |
| **Encryption** | Scrambling data to make it illegible |
| **Encryption algorithm** | Defines the rules by which data will be transformed so that it becomes illegible |
| **Encryption at rest** | Protecting data while it is stored |
| **Encryption in transit** | Protecting data while it is transmitted from one location to another |
| **Encryption in use** | Protecting data when it is in use in memory |
| **Identity and access management** | Also known as access control, it helps authenticate and authorize users and provides user-specific access to cloud resources, services, and applications |
| **Infrastructure monitoring tools** | Identify minor and large-scale hardware failures and security gaps so that developers and administrators can take corrective action before problems affect user experience |
| **Key management services** | Help perform life cycle management for encryption keys that are used in cloud services or customer-build apps |
| **KYOK** | Keep Your Own Keys |
| **Multifactor authentication** | Adds an additional layer or authentication for application users |
| **Reporting** | Provides a user-centric view of access to resources |
| **Server-side encryption** | Occurs after cloud storage receives your data but before the data is written to disk and stored |
| **SSL** | Secure Sockets Layer |
| **TLS** | Transport Layer Security |
| **User and service access management capability** | Enables cloud application and service owners to provision and de-provision user profiles with minimal human interaction |

**Module 2 Glossary: HTML5 Overview**

| **Term** | **Definition** |
| --- | --- |
| **DOM Tree** | “Document Object Model” is the data representation of the objects that comprise the structure and content of a document on the web. |
| **XML** | An “eXtensible Markup Language” Designed to store and transport data allowing users to define their own markup languages, primarily to displaydocuments on the web. |
| **XHTML** | An “eXtensible Hypertext Markup Language” similar to HTML but with stricter formatting rules. |
| **Web Storage APIs** | APIs that allow data storage in a browser. |

**Module 3 Glossary: CSS Overview & HTML5 Elements**

| **Term** | **Definition** |
| --- | --- |
| **Component Framework** | Component frameworks provide pre-styled components and templates which are easy to add to any website. |
| **CSS** | "Cascading Style Sheet"s is a style sheet language that describes how HTML elements are displayed​. It is the design that is layered over the top of an HTML web page​. |
| **Fixed Layout** | A fixed layout is a layout where ​you specify the height and width of elements, and those values remain the same regardless of which operating system or browser you use to access the website. |
| **Fluid Layout** | A fluid layout is a layout in which ​the height and width of elements is flexible ​and can expand or contract based on the browser window, the operating system, and other user preferences. |
| **Utility Framework** | The utility framework provides utility classes that are scoped to individual CSS properties, which helps in building custom designs in HTML files. |

**Module 4 Glossary: JavaScript Programming for Web Applications**

| **Term** | **Definition** |
| --- | --- |
| **AJAX** | “Asynchronous JavaScript and XML” that encompasses more than asynchronous server calls through JavaScript and XML. It is not programing language or technology but rather a programming concept. Ajax represents a series of techniques that provide richer, interactive web applications through HTML, JavaScript, Cascading style sheets, and modifying the web page through the Document Object Model. The name is misleading though because nowadays, JSON is commonly used instead of XML. |
| **Anonymous Functions** | A type of function that has no name or we can say which is without any name. It is declared without any identifier and is often used as a parameter of another function. It is a common way to execute a function immediately after its declaration. |
| **Array** | A data structure that aids the programmer in the storage and retrieval of data by indexed keys. Arrays use a zero-based indexing scheme, meaning that the first element of an array has an index of zero. Arrays grow or shrink dynamically by adding or removing elements. |
| **Classes** | Classes act as a blueprint or template for building objects with similar characteristics and behaviours. A class encapsulates data (in the form of properties) and functions (in the form of methods) that work on that data. |
| **Client-Side Script** | A program that accompanies an HTML doc or embedded in HTML. Scripts run during load of a document or when an action is performed. They can be used to validate forms, process input, or dynamically create document elements. Embed a script in HTML, with the <script> tag in either of the following ways:   • <script> // JS code </script>   • <script src="path name"></script> Use <noscript> tag for browsers with JavaScript disabled or ones that don’t support JavaScript. |
| **Document Objects** | Document representing the main web page that gives access to all HTML elements on the page. When page is loaded HTML doc becomes a document. It is referred to with “document”. |
| **DOM** | “Document Object Model” is a programming interface (API) between HTML and JavaScript. It allows for dynamically accessing and updating content, structure, and style. JavaScript uses the DOM to access and modify web page elements in the browser. |
| **Element Nodes** | All HTML tags. |
| **Element Objects** | The most general base class that all element objects in a Document inherit. It only has methods and properties common to all elements. Everything in a HTML page is an element. And one element can have other elements nested within itself. |
| **Event** | An event is something either a browser or a user does that the JavaScript can react to such as a button click or when a user submits input on a form. |
| **Event Binding** | Refers to telling the browser that a function should be called whenever some ‘event’ occurs. |
| **Event Handlers** | A function that declares what to do when an action is performed such as the click of a button. Example: <button type="button" onclick="showAnswers()"> Show Solution  <script>   function showAnswers() {    //code    alert("A")   }  </script> </button> Note that showAnswers() is an event handler. |
| **Extend** | This keyword is used in class declarations or class expressions to create a class that **is** a child of another class. |
| **Functions** | Functions are modules of code that execute a particular task. They may take-in data, called arguments or parameters, and sometimes return data as well, called the return value. Functions are commonly defined with this syntax: function functionName() {  // function code;  // optional return statement; } |
| **IIFE** | “Immediately Invoked Function Expression” runs immediately after it is defined.  After the function executes it cannot be called again elsewhere in the program.  It is a type of self-executing function. |
| **Nodes** | The basis of all elements in the Document Object Model (DOM) structure. |
| **Objects** | Objects are instances created from a class. They are real-world entities that represent the characteristics defined by the class. Objects have a special set of properties that store data and methods that specify behaviours. These methods and properties can be accessed and changed to carry out specific tasks and communicate with other programs. |
| **Prototypes** | A function prototype lets you easily define and add properties or methods to an object. Prototypes exist for all objects that can be created with the keyword”new”. All object constructors create objects that inherit properties and methods that are defined by the prototype. At instantiation objects inherit the current state of the prototype. Note however, that scripts can override prototype properties and functions. Following is an example of using a prototype to add a method to the Car class: function Car(make, model, year) {  this.make = make;  this.model = model;  this.year = year; } Car.prototype.getName = function() {  return this.make + ‘ ’ + this.model + ‘ ’ + this.year; } |
| **Script** | Offers developers  means to modify and extend HTML documents in highly interactive ways. Scripts can be used to validate forms or to process input as it is typed. Scripts can be triggered by events that occur on a web page, such as the clicking of a button. Scripts can be used to dynamically create document elements on an HTML page. |
| **Self-Executing Functions** | Often used to initialize data or declare DOM elements.  These functions can be  anonymous. |
| **Text Nodes** | The nodes that contain actual text that go between an element start tag and end tag. |
| **this** | Keyword “this” refers to current instance of the object. The value of “this” can vary depending on how the object is called. |