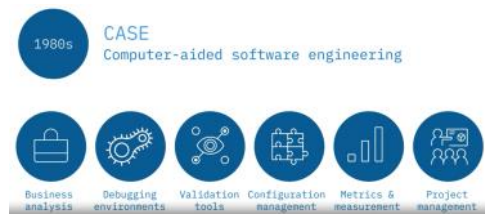


What is Software Engineering?

17 September 2024 04:08 PM

Explaining about organized software application for consistent performance.

CASE :



Software Developer vs Software Engineers:



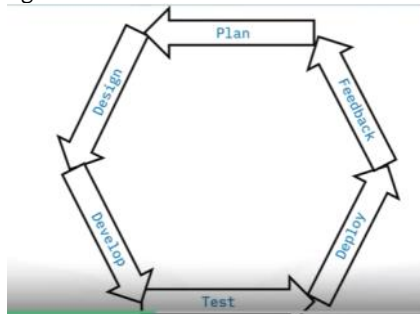
Software Engineers responsibilities:

1. Design, build, maintain sw systems
2. Write & test code
3. Consult with stake holders, third party vendors, security specialists and other team members.

Software Development Life Cycle (SDLC):

1. Scientific approach to sw development.
2. Guides the sw development process.
3. Identifies discrete steps needed to develop software.

Agile:

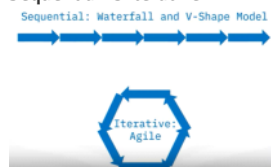


1. collaborative software development process over multiple short cycles rather than a strictly top-down linear process.
2. Iterative approach to development. Phases is short
3. sprints, which are usually one to four weeks long
4. Unit testing happens in each sprint to minimize the risk of failure
5. Rather than the “maintenance” stage of the SDLC, the final stage of the sprint is a feedback stage
6. working code is released at a meeting called the “sprint demo”
7. After several sprint cycles, a minimum viable product, or MVP(provides assumptions about the software.), is developed

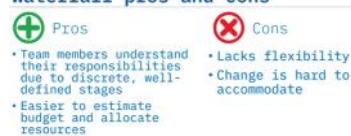
Core values of agile called 'Agile Manifesto':

1. individuals and interactions over processes and tools
2. working software over comprehensive documentation
3. customer collaboration over contract negotiation
4. responding to change over following a plan.

Sequential vs Iterative:



Waterfall pros and cons



V-shape model pros and cons



Pros

- Easy to use
- Test plans designed upfront saves development and testing time



Cons

- Rigid
- Does not accommodate changing requirements

Agile pros and cons



Pros

- Changing requirements handled easily
- Feedback incorporated regularly



Cons

- Budgeting and resource allocation is challenging
- Project scope not clearly defined

encouraged to provide feedback - As coding languages and technologies have developed in recent years, they now allow for modular design, where developers can focus on smaller chunks of code that are readily integrated into the larger product. These small chunks can be released to provide the MVP.

Recap

In this video, you learned that:

- Three common development methodologies are waterfall, V-shape model, agile
- Waterfall and V-shape are sequential while agile is iterative
- Waterfall and V-shape models are easy to use but neither accommodates changing requirements well
- Agile allows for changing requirements but resource allocation can be challenging

Software Versions:

1. Software Versions are identified using Software Versions numbers
2. Information with Software Version numbers.

Software versions

- Software versions are identified by version numbers
- Version numbers indicate:
 - When the software was released
 - When it was updated
 - If any minor changes or fixes were made to the software

3. Developers can keep track of new software updates and patches.
4. V.number can be short/ long
5. Each num set is divided by period
6. Note: A version still in beta or testing could have a version number lower than 1, such as 0.9.
7. Some uses dates for versioning such as ubuntu linux

What do version numbers mean?

Some version numbers follow the semantic numbering system and have 4 parts separated by a period

- The first number indicates major changes to the software, such as a new release
- The second number indicates that minor changes were made to a piece of software
- The third number in the version number indicates patches or minor bug fixes
- The fourth number indicates build numbers, build dates, and less significant changes

About

version 9.1.33.6

- Older versions may not work as well in newer versions
- Compatibility with old and new versions of software is a common problem
- Troubleshoot compatibility issues by viewing the software version
- Update software to a newer version that is compatible
- Backwards-compatible software functions properly with older versions of files, programs, and systems

Testing:

1. Software Testing is the practice of Integrating quality checks throughout the software development cycle.
2. Purpose of testing - is to check whether the software matches expected requirements and ensure error-free software.

Test software,

1. the team writes "test cases."
2. Test cases can be written in different stages of the SDLC.

A test case contains:

- 1.steps,
2. inputs,
3. data,
5. expected corresponding outputs.

Categories:

1. Functional testing
 1. testing without looking at source code or internal structure
 2. concerned with inputs and corresponding outputs of the system under test(blackbox testing) called SUT
 3. user errors or input edge cases do occur, the software handles those exceptions seamlessly by displaying appropriate error messages.
2. Non-Functional testing
 1. testing the application for attributes like performance, security, scalability, and availability. Non-functional testing checks to see if the SUTs non-functional behavior is performing properly.

2. How does the application behave under stress? What happens when many users log in at the same time? Are the instructions in documents and user manuals consistent with the application's behavior? Does the application behave similarly under different operating systems? How does the application handle disaster recovery? And how secure is the application?

3. Regression testing

1. 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.

2. test case selection and prioritization can be challenging and can depend on several factors. include cases that: have frequent defects , contain frequently used functionality, contain features with recent changes, or are complex test cases, edge cases, and randomly successful or failed test cases.

There are four testing levels:

1. unit - testing functionality of a specific section of code by developers/engineer during the development phase aims to eliminate construction errors

2. integration- seeks to identify errors when two or more smaller, independent code modules are combined. passed unit testing are incorporated into the larger software application. bugs that occur

when those smaller units of code interact with each other. uncovers deficiencies in communication with a new module in conjunction with other existing modules, databases, or external hardware. Poor exception handling can cause problems when modules are integrated

3. system - It validates the system as a fully completed software product in a staging environment(similar to the production environment) after integration testing being conducted . System testing is both functional and non-functional.

4. acceptance- It determines whether a system

satisfies the needs of the users, customers, and other stakeholders by the customer or the stakeholders during the maintenance stage of the SDLC.

Recap

In this video, you learned that:

- There are three categories of testing: functional, non-functional, and regression
- Unit testing verifies small, independent chunks of code
- Integration testing looks for errors when two or more small chunks of code are combined
- System testing validates the system as a fully completed product
- Acceptance testing verifies implementation of user requirements and business processes

Responsibilities role in Software Engineering:

different names depending on the approach being used, such as Agile or waterfall.

Software development project roles

- | | |
|-----------------------------------|---|
| 1. Project manager / Scrum master | 6. Tester / QA engineer |
| 2. Stakeholder | 7. Site reliability / Ops engineer |
| 3. System / Software architect | 8. Product manager / Product owner |
| 4. UX Designer | 9. Technical writer / Information developer |
| 5. Developer | |

Responsibilities of Each Role:

Project manager / Scrum master

- | Project manager | Scrum master |
|---|--|
| <ul style="list-style-type: none">• Planning, scheduling, and budgeting• Allocating personnel and resources• Executing the software plan• Team communication | <ul style="list-style-type: none">• Ensure team success• Individual success• Prioritize people over process• Focused on facilitating communication between all team members |

Stakeholders:

1. people for whom the product is being designed.

2. They include individuals such as the

- customer,
- end-users,
- decision-makers,
- system administrators,
- other key personnel.

Responsibilities:

1. Responsible for defining project requirements

2. Providing feedback if the team members need clarification on requirements or if a proposed solution cannot be solved as planned.

3. The stakeholders may also sometimes participate

in beta testing and acceptance testing before the software is released.

System Architect:

Called as Software architect /solution architect.

System architect

Designs, describes, and communicates architecture of a project to team members

- 
- Responsibilities:
- Designs inner structure
 - Designs technical aspects
 - Provides technical support regarding the architecture

UX designer

Balances making software interface intuitive yet also robust



Responsibilities:

- How software communicates functionality to the user
- How the user interacts with the software

Developer

Writes the code that powers the software



Responsibilities:

- Implement architecture in design documents
- Incorporate SRS requirements
- Employ UX design

Tester / QA engineer

Ensures quality of the product by testing to see if the software meets requirements



Responsibilities:

- Writes and execute test cases
- Provides feedback to development teams

Site reliability engineer

Bridges software development and operations



Responsibilities:

- Tracks and communicates incidents
- Automates systems, procedures, and processes
- Trouble shoot
- Ensures product reliability

assist with trouble shooting;
ensure reliability for the customer.

The product manager or 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.

Product manager / Product owner

Understands requirements and end-user needs



Responsibilities:

- Leads development efforts
- Ensures product provides value to the customer

Technical writer or Information developer :

- writes documentation for the end-user.
- They write documentation on technical material geared towards a non-technical audience.
- Not only does this documentation help the end-user to use the software, but it also helps the customer so they can provide timely feedback to the development teams.

Technical writer / Information developer

Writes documentation for end-user

Technical material → Non-technical audience



Responsibilities:

- Writes user manuals
- Writes reports
- Writes white papers
- Writes press releases

Web and Cloud Development.

1. What you need to learn and what order you should learn it in.
2. Understanding how familiar websites are constructed and delivered to you

basics of how you interact with a website.

1. You launch an internet browser – there are lots available: Google Chrome, Microsoft Edge, Mozilla Firefox, and Apple Safari are some of the most popular.
2. The browser has an address bar, into which you enter a URL, like www.ibm.com.
3. The browser then contacts the server with the name and requests the information. The server then sends a response, which contains the data that the client requires to display the website.
4. For most websites, the server will return: HTML, which defines the structure of the page, but doesn't look very attractive CSS, which adds style and flair to the page and JavaScript, which adds interactivity and dynamic content.
5. Content displayed either static or dynamic.
6. Dynamic elements can involve information coming from other systems and applications, such as databases.
7. Most websites contain static and dynamic elements to provide the best user experience.

Cloud Applications:

Cloud applications



- Built to work seamlessly with a Cloud-based back-end infrastructure
- Cloud-based data storage and data processing, and other Cloud services, making them scalable and resilient

Cloud Applications - primary areas:

1. Front-End :

Building websites and cloud applications

The environment is divided into two primary areas:



Front-End

- Deals with everything that happens at the client-side
- Specializes in front-end coding, using HTML, CSS, JavaScript and related frameworks, libraries, and tools

2. Back-End:

Building websites and cloud applications

The environment is divided into two primary areas:



Back-End

- Deals with the server before the code and data are sent to the client
- Handles the logic and functionality and the authentication processes that keep data secure
- Back-end developers may also work with relational or NoSQL databases

Building websites and cloud applications:

1. Full-stack developers have skills, knowledge, and experience in both front-end and back-end environments.
2. appropriate Developer tools to help:

Developer tools



First tool most developers add to their resources is a code editor



IDE integrates with management and storage tools like Git and GitHub



Integrated Development Environment or IDE helps to build and manage codes

Developer tools

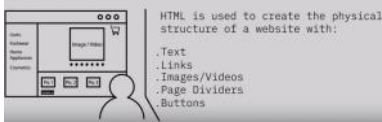
Examples for code editors and IDEs include:

3.
 - Sublime Text
 - Atom
 - Vim
 - VS Code
 - Visual Studio
 - Eclipse
 - NetBeans

Different services of Websites:

Front-End Developer:

To create a website, web developers usually use Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. These languages are designed to work in conjunction with each other.

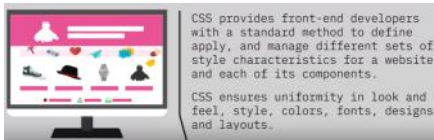


The HTML code ensures a proper formatting of all text and image elements so that browsers display the page consistently.

Backend Developer:



CSS:




A front-end development language is Syntactically Awesome Style Sheets (SASS)



An extension of CSS that is compatible with all versions of CSS.

SASS enables you to use things like variables, nested rules, inline imports to keep things organized.

Learner Style Sheets (LESS)




LESS enhances CSS, adding more styles and functions.

It is backwards compatible with CSS.

Less.js is a JavaScript tool that converts the LESS styles to CSS styles.

Reactive Websites:

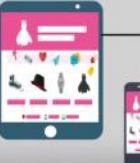
Websites are designed as reactive and responsive



Reactive or adaptive websites display the version of the website designed for a specific screen size.

A website can provide more information if opened on a PC than when opened on a mobile device.

Responsive Websites:



Responsive design of a website means that it will automatically resize to the device.

If you open up a products website on your mobile device, it will adapt itself to the small size of the screen and still show you all the features.

Java script frameworks:

Angular :

Angular framework is an open-source framework maintained by Google.

Angular framework allows websites to render HTML pages quickly and efficiently.

It has built-in tools for routing and form validation.

React :

As Angular JS the routing is not part of the framework. It helps to build and drop component into a page.

React.js has been developed and maintained by Facebook.

It is a JavaScript library that builds and renders components for a web page.

Routing is not a part of this framework and will need to be added using a third-party tool.

Vue:

Vue.js is maintained by the community and its main focus is the view layer which includes user interface, buttons, visual components.

It is flexible, scalable and integrates well with other frameworks. It is very adaptable - it can be a library, or it can be the framework.


Backend Development:

Developer roles



Front-end developer

- Creates websites and Cloud Apps that the user interacts with



Back-end developer

- Creates and manages resources needed to respond to client requests
- Enables server infrastructure to process request, supply data and provide other services securely

Both work together closely throughout the life of the website or Cloud App to resolve issues and add functionality

Examples of tasks and associated skills that back-end developers need:

Search requests submitted to database	Navigation to restricted areas	Secure payments
Understand web application development language	Provide access based on user's log in status	Secure data handling of sensitive information
Request database for correct data	Manage, authenticate and authorize user account	Secure storage of sensitive information

Working with Databases:

1. SQL
2. ORM - object Relational Mapping
 1. ORM can hide some of the complexity of querying databases so need to know the basics of databases to troubleshoot any issues

- Back-end developers focus on the functionality that keeps websites, cloud apps and mobile apps up and running
- Back-end development covers a wide range of technologies:
 - Managing user accounts
 - Authentication and authorization
 - Secure storage and handling of sensitive data
- Back-end developers work with databases, retrieving, processing and storing data

Teamwork and Squads :

Teamwork:

Group of people working together towards a common aim.

Including different people with different skills, experience and talents.

Promotes creativity - Collaborating with others gives you the opportunity to discuss ideas and challenge one another's thinking about a subject.

Empowering - positive attitudes and behaviors can impact the rest of the team and create positive results.

How to work well as a team:

Trust and respect - needs to trust and respect the other members of the team. Generally comes with time and contributing equitably.

Define and confirm goals - whole team knows what they are working towards.

Define and confirm roles - need to define and agree on roles to avoid any duplication of effort or missed tasks.

Working with each members strengths - Working with each members strengths is important to make the most of the talent within your team, as is celebrating success and analyzing problems.

Communication - choose a method that works for everyone so that the whole team is seeing and responding to information.

Teamwork in software engineering:

Kickoff- They plan how they will complete the project, assign tasks, and agree on goals.

Design and code Reviews - code reviews can be requested at the team level .

Walkthroughs - Team members might present walkthroughs of their sections of responsibility to the rest of team so that the whole team has oversight of all parts of the project.

Retrospective - meetings may be held to review what went well and what could be improved in future projects.

Mentor - mentor who may or may not be in your current project team.

- Some groups also have teams working on internal projects such as defining code standards, maintaining or updating legacy cross-project code, or reviewing potential new software for usefulness to the team.

Benefits of teamwork

- Encourage creativity
- Take advantage of strengths
- Gain knowledge and skills from others
- Adhere to standards and document code
- Better quality code and fewer bugs
- Less stress
- Everyone sees the bigger picture

Squads:

1. Agile development methodologies may call a team a squad.
 - Squad leader** - the anchor developer and coach for the squad.
 - Product developers** - developers
 - UX developers and designers - designers
 - Work together with pair programming.

Pair Programming:

Pair programming is a type of Agile development where two developers can plan and discuss their ideas continually as they create a solution, generally resulting in a better end product.

- Two developers, one computer
 - Physically side-by-side
 - Virtually via video or shared screens

Styles of Pair Programming:

- Driver/navigator
 - "Driver" types in the code
 - "Navigator" reviews code as it's written
 - Important to swap roles regularly
- Ping-pong
 - First developer writes a test
 - Second developer writes code to pass the test
 - Important to swap roles regularly
- Strong style
 - Pair a junior programmer with an experienced one
 - Senior programmer is the navigator
 - Junior programmer is the driver

Benefits of pair programming

- Share knowledge and skills
- Builds soft skills
- Fewer typos, logic errors, and bugs
- Code review on the fly
- Identify optimal solution early on



Challenges of pair programming

- Exhausting
- Personal and work commitments impacting schedule
- One person taking control
- Personality clashes
- Noisy environment



Application development Tools:

Getting Cloud App from the ideas stage to fully formed, written, and deployed is a long process, but there are many tools to help.

Cloud application developer's workbench includes:

- Version control
- Libraries
- Frameworks

Version Control :

As many developers working on same project to know the order changes is important.

1. It keeps track of:
 - What changes were made
 - When they were made
 - Who made them
2. when and by whom and resolve any conflicts between changes
3. Provides away to revert to an older version. Generally tied to the storage system.

Git & Github - stores files in repositories 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.

Libraries:

- Collections of reusable code, like standard programs and subroutines
- Multiple code libraries can be integrated into your existing project
- Required method is called when it is needed.
- The control returns to the program flow once the subroutine is finished.
- Used to solve a specific problem or add a specific feature set

Examples of code libraries are:

- jQuery is a JavaScript library that simplifies DOM manipulation
- Email-validator checks if email address is correctly constructed and valid
- Apache Commons Proper is a repository of reusable Java components

Frameworks:

- Provides a standard way to build and deploy applications
- Acts as a skeleton we can extend by adding our own code
- Should be determined and used from the beginning of the project
- The framework determines which subroutines and methods will be called when. Frameworks defines the workflow you must follow
- Specific structure must be follow
- AngularJS - dynamic Web applications
- Vue.js - Focused on the user interface
- Django - mainly used for Web development

CI/CD:

- Continuous Integration / Continuous Deployment
- Enable developers to deliver frequent changes reliably
- Implemented through a build-automation server. It builds and tests the code then CD deploys the code changes in a testing / staging environment.

Build Tools:

- Transforms the code into binaries needed for installation
- Organizes the code, set compile flags and manage dependencies.
- Useful when many developers works in same project.

Automate tasks like:

- Downloading dependencies
- Compiling source code into binary code
- Packaging that binary code
- Running tests
- Deployment to production systems

2 categories of build tools:

Packages :

Package managers:

Interpreted and compiled programming languages:

1. Identify interpreted programming languages,
2. identify compiled programming languages.

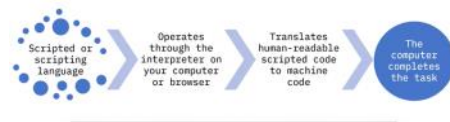
What are programming languages?

- Programming languages help us tell computers what to do.
- human-readable programming languages. -> Computers language machine code - binary code

categories for programming languages:

Interpreted and compiled

Interpreted programming:



- Interpreted programming languages need almost any OS with an right interpreter to translate the source code.
- Translators are built into your web browser or they require a program on your computer to translate the code.
- Scripted source code(smaller & repeated tasks) -> Goes through an interpreter -> built into your web browser / OS
- Used to automate tasks
- Execute code line by line
- Source code should be executed each time

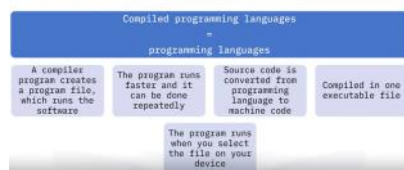
Interpreted programming examples



Compiled Programming Languages:

- Compiled programs are applications and programs, like your music app or your operating system, that you run on your computer or device.
- The programs are packaged -or compiled- group into one downloadable / into one executable file.
- They are usually larger programs. Compiled programs are used to help solve more challenging problems, like interpreting source code.
- Your operating system is written in a compiled programming language, like C, C++, C#, or Java requires JVM to run code and it is a programming language for AndroidOS .
- Same OS and Same Language

Compiled programming



Examples of compiled programming languages are:

- C, C++ and C# are used in many operating systems, like Microsoft Windows, Apple macOS, and Linux
- Java works well across platforms, like the Android OS

Process of compilation:

1. When you update to a new version of your operating system, like Microsoft Windows, your device might download an installation program.
2. That program is made up of many files. The files are written in a compiled programming language.
3. These files give instructions to your device in machine code.
4. The compiled program is running on your device. A compiled program that you commonly use is your device's operating system, such as Linux, Microsoft Windows, Apple's macOS, or Android.

Recap

In this video, you learned that:

- Interpreted programming languages run scripts that are repetitive and need to be run often
- Interpreted programming languages are more versatile and can be used across platforms as long as there is the correct interpreter
- Some examples of interpreted programming languages are JavaScript, Python, and HTML
- Compiled programming languages are for more complex programs that complete larger tasks
- Compiled programming languages are used for creating executable files that can run directly from your device
- Some examples of compiled programming languages are C and Java

How developers choose programming Language:

1. Depends they have experience and trust.
2. Best for users
3. Most efficient to use

Recap

In this video, you learned that:

- Interpreted programming languages create source code that runs through an interpreter in your device's operating system or on your web browser
- Compiled programming languages create executable files that are grouped in programs on your device
- Compiled programming languages like C and Java are used to write larger programs like operating systems and other executable files
- Interpreted programming languages like Python and HTML are used to write code that can complete repetitive tasks within a web browser or a computer

Query and Assembly programming languages:

Programming language levels:

High level:

1. Uses common English
2. More sophisticated
3. SQL, Pascal, Python

Low level:

1. Uses simple symbols to represent machine code
2. ARM, MIPS, X86

Query Language:

- It is predefined and understandable instructions
- Query is a request for info from a database. The data base searches the table and returns the result.
- User application and the database handling the query are speaking the same language.
- Query Languages : AQL, CQL, Datalog and DMX
- Used to do CRUD operations

When a user performs a query, the database:

1. Retrieves data from the table
2. Arranges data into some sort of order
3. Returns and presents query results

SQL VS NoSQL:

NoSQL - Not Only SQL:

1. Data structure is different from the SQL db's
- SQL databases:**
- Relational
 - Use structures, predefined schemas.
- NoSQL Databases:**
- Non-relational
 - Dynamic schemas for unstructured data

SQL Statement	Task
SELECT	Extract data from a database. All request queries start with a SELECT statement.
WHERE	Filter the query criteria to return results that match a set condition (used in combination with SELECT).
ORDER BY	Define the sort order of returned results.
UPDATE	Modify existing data in a database.
INSERT INTO	Add new data records to a database.
ALTER TABLE	Add or remove columns in a database.
CREATE	Create a new object in a database. Object options are DATABASE, TABLE, INDEX, and VIEW.
DROP	Delete an object from a database. Object options are the same as above.
SUM	Return total sum of a column of numeric data.
COUNT	Return number of rows that match a condition.
AVERAGE	Return average value of a column of numeric data.

Assembly Language:

- Less sophisticated than query languages than, structured programming languages and OOP languages.
- Low level programming language
- Uses simple symbols to represent os and 1s
- Closely tied to CPU as every CPU have its own assembly language.
- Translated using assembler using mnemonics
- One statement translated into one machine code instruction

Syntax:

Simple readable and one line/statement at a time

```
{label} mnemonic {operand list} {;comment}
```

```
mov TOTAL, 212 ; Transfer the value 212 in
                ; the memory variable TOTAL
```

Translate using mnemonics:

- Input (INP), Output (OUT), Load (LDA), Store (STA), Add (ADD)

Statements consist of:

- Opcodes that tell CPU what to do with data

Recap

In this video, you learned that:

- Query languages, structured programming languages, and object-oriented programming languages are categorized as high-level programming languages
- Assembly languages are categorized as low-level programming languages
- A query language is predominantly used to request data from a database or to manipulate data in a database
- The most prevalent query language for database queries and database management is Structured Query Language (SQL)
- Select queries request data from a database, whereas action queries manipulate data in a database

Recap

In this video, you learned that:

- Assembly languages use a simple set of symbols to represent the 0s and 1s of machine code
- Assembly languages are closely tied to the processor architecture from hardware manufacturers
- Assembly languages are translated using an assembler instead of a compiler or interpreter
- Assembly language instructions have a one-to-one association with their machine code counterpart

Importance of Code organization:

- Readability
- Maintainability
- Configuring

Planning and organizing software design:

- Enables writing cleaner more reliable code
- Helps improve code base
- Reduces bugs and errors
- Has a positive impact on program quality
- Provides consistent and logical format while coding

Pseudocode



- Informal, high-level algorithm description
- Step-by-step sequence of solving a problem

Flowcharts



- Pictorial representation of algorithm, displays steps as boxes and arrows
- Used in designing or documenting a process or program

Pseudocode



- Bridge to project code; follows logic
- Helps programmers share ideas without extraneous waste of creating code
- Provides structure that is not dependent on a programming language

Flowcharts



- Good for smaller concepts and problems
- Provide easy method of communication about logic behind concept
- Offer good starting point for project

Flow charts:

- Pictorial representation of an algorithm
- Having standard symbols
- Common flow chart softwares are, Microsoft visio, Lucidchart, Draw.io, DrawAnywhere

Start

Process

Decision

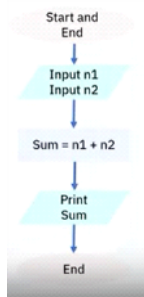
Data

Connector



Loop

A loop represents a flow that iterates under a set of conditions.



Pseudocode:

- Developers use pseudocode to know requirements and align code accordingly
- Summarize program flow but excludes underlying details
- Code executor provides ability to plan instructions that follow a logical pattern
- Changes in the design can be easily incorporated

Recap

In this video, you learned that:

- Organizing and planning software before development is crucial to good code development
- Organized code is easier to read, maintain, and configure
- Flowcharts and pseudocode are two methods of planning and organizing code before development
- Flowcharts display a process, represented using different shapes and arrows
- Pseudocode explains exactly what each line does, making it easier for programmers to understand

Branching and Looping:

In this video, you learned that:

- There are two types of programming logic: branching and looping
- Boolean expressions and variables are different but also part of programming logic
- Branching is deciding what actions to take; looping is deciding how many times to perform an action

Programming structures:

Recap

In this video, you learned that:

- Software developers use identifiers to reference a program component
- Data identifiers can either be a constant or a variable
- A constant is a data item whose value does not change
- A variable is not constant; it can change during the program's execution
- Arrays have a fixed number of elements stored in sequential order

Functions:

- Modular programming

Types of functions:

1. Standard library functions
 - Interpreted programming languages create source code that runs through an interpreter and is built into your operating system (OS) on your computer or on your web browser.
- Compiled programming languages create executable files that are grouped in programs on your computer or device.
- Query languages, structured programming languages, and object-oriented programming languages are categorized as high-level programming languages and assembly languages are categorized as low-level programming languages.
- The two main methods of organizing and planning code are by developing flowcharts and by writing pseudocode. Flowcharts are pictorial representations of algorithms and pseudocode is an explanation of the function of each line of a program.
- To reference a program component, software developers use an identifier, which can either be a constant or a variable.
- A function is a piece of structured, stand-alone, and reusable code that will perform a single specific action.
- Object-oriented programming is a programming methodology that is focused on objects rather than functions.

Recap

In this video, you learned that:

- A function is stand-alone, reusable code that will perform a single action
- Defining and calling functions is common to all programming languages
- OOP is a programming methodology that is focused on objects
- Software objects consist of properties and methods

Software Architecture:

- Software design and documentation take place during the design phase of the SDLC.
- Software architecture, simply put, is the organization of the system.
- 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.

Early Design decisions:

1. How the components interact among them.
2. Operating environment.
3. Design principles
4. Costly to change once implemented.
5. Addresses non-functional aspects of the software such as performance, scalability, maintainability, interoperability, security and manageability

Why SW arch is important:

1. Different need of the stakeholders is the basis for communication among team members.
2. Flexibility to change in requirements
3. Increases the life plan

Software architecture and tech stacks:

- Guides to use of the technology system / stack choice.
- Tech stack must address non-functional capabilities.
Tech stack includes:
 - software
 - programming languages
 - Libraries
 - Frameworks
- Architect must know the advantages and disadvantages of tech stack choices.

Artifacts:

Several artifacts produced during the architectural design phase. This includes,

1. Software design document
2. Architectural diagram
3. UML diagrams

Software Design Document:

- Collection of technical specifications that indicate how the design should be implemented.
- Design Considerations:
 - Assumptions
 - Dependencies
 - Constraints
 - Requirements
 - Objectives
 - Methodologies

Architectural Diagrams:

Architectural diagram displays,

- Components
- Their interactions
- Their constraints
- Their confines
- It displays the architectural patterns used in the design. They are general reusable solutions to commonly occurring problems.

UML Diagrams:

UML diagrams visually communicate structures and behaviors using common programming.

Deployment Consideration:

- Software architecture includes production deployment considerations.
- The architecture drives choices about the environment in which the software is released.
- 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.

Recap

In this video, you learned that:

- Software architecture functions as a blueprint and represents the underlying organization of the application
- Software architecture is important because it serves as a basis for communication among team members
- It represents the earliest design decisions, is hard to change once development starts, and accommodates changing requirements during development

Software Design:

Software design is a document before the software is being developed. It consists of

1. Structural components
2. Behavioral components

Modeling the software to express its design of different components and the interactions between them:

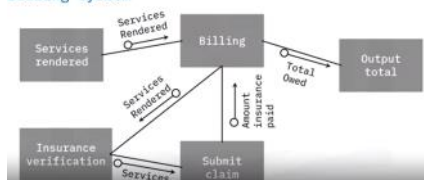
1. Diagrams and Flowcharts
2. UML diagrams

Structural Design:

- It can be construed in terms of structural elements
- The software problem into small modules and sub modules
- This design should contain modules that are cohesive and loosely coupled.
- Cohesion - all functionality related elements are grouped together
- Loosely coupled - Coupling means communication between the modules. The modules should be weakly coupled so changes in one component have minimal effect.

Structure diagram example

Billing system



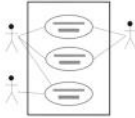
- Billing is the module
- Service rendered, Insurance verification, submit claim, output total are sub modules
- Arrows represents flow of data.

Behavioral models:

- Behavioral models explains what system does rather than explaining how it does.
- It describes the relationship between the objects/components
- Types of behavioral models:
 - State transition diagram
 - Interaction Diagram

UML Diagrams

- Visual representations to communicate architecture, design, and implementation
- Two types: structural and behavioral
- Programming language agnostic

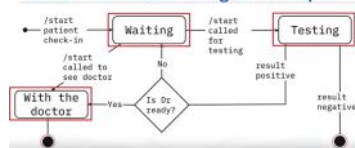


- Uses of using UML diagrams:



State transition diagram:

State transition diagram example



- Rectangular boxes are states

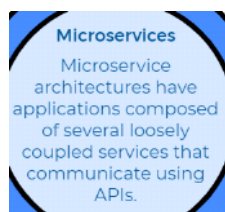
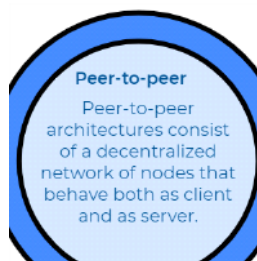
Interaction diagram :

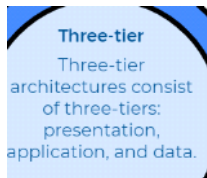
- It is used to represent dynamic nature of the software system.
- It describes the relationship between the objects/components
- Sequence diagram - type of interaction diagram to visualize objects & relationship

Recap

In this video, you learned that:

- Structured design breaks down a software problem into well-organized solution elements
- Behavioral models describe the behavior of a system
- UML diagrams acclimate developers to the project, plan features, navigate source code
- A state transition diagram describes different states of a system and triggering events
- An interaction diagram describes how interacting objects communicate





event-driven. Other architectural patterns include model-view-controller, blackboard, message-broker, pipe-filter, and controller-responder. To continue, click **Next**.

Application Architecture:

Component:

- An individual unit of encapsulated functionality, part of application in conjunction with other components
- Characteristics of components:
1. Reusable
 2. Replaceable
 3. Independent - Does not have dependencies
 4. Extensible - Add behavior without changing other components
 5. Encapsulated - hide specific implementation
 6. Non-context specific - Operates in different environment

Examples of components:

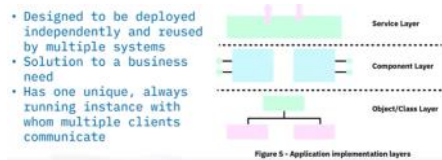
API - can be packaged as component, reuse across multiple systems/apps

Data Access object - it's a interface for Database called data access object switches the user to a different database without the application knowing about the switch.

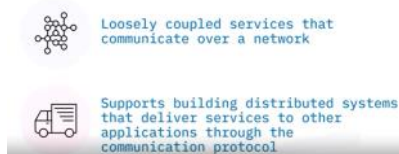
Controller - determines which component need to be called for which event. Controls the flow of data between 2 components.

Component-based architecture:

- Decomposes design into logical components
 - Higher level abstraction than objects
 - Define, compose Loosely coupled independent component
- Services



Service-oriented architecture



What is distributed systems:

Distributed systems



Characteristics of distributed system:

- Shares resources
- Fault-tolerant
- Multiple activities run concurrently
- Scalable
- Runs on a variety of computers
- Programmed in a variety of languages

Nodes:

- Any device on a network is called node, that can recognize, process and transmit data to other nodes on the network.

DS architectures :

1. Client- server
2. Peer-to-peer
3. 3-Tier
4. Microservices

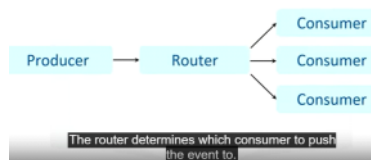
Recap

In this video, you learned that:

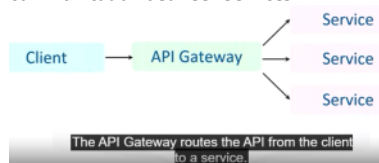
- Components are reusable, independent, replaceable, extensible, encapsulated, and non-context specific
- Component-based architecture is the decomposition of the system into logical independent components
- Services are made of components and components consist of objects. Services are deployed independently and can be reused by multiple systems
- SOA is a loosely coupled services that interface with each other via a communication protocol over a network
- Distributed systems:
 - multiple services on different machines

Architectural patterns in software:

- Repeated solution to a problem
- **2-tier** - client server model, client make request for data / other services to the server
- **3-tier** - consist of presentation Tier, Application Tier and Data Tier. N-tier .. Several horizontal tiers that function together as single unit of software. Related components are places within the same tier. Changes in one tier do not affect other tier.
- **Peer-to-peer** - Decentralized network. Each node works as both client and server. Workload partitioned among these nodes. Resources - processing power, disk storage, network bandwidth. Useful for file sharing, instant messaging, collaboration and high performance computing.
- **Event-driven** - results in a change of state such as mouse click. Focuses on producers and consumers. Producers - listen for and react to trigger. Consumer- process an event .



- **Microservices** - breaks application into functionality called services. An API defines how 2 applications share and modify each other's data. Orchestration handles communication between services.



- Model-view-controller
- Message-broker
- Blackboard
- Pipe-filter
- Controller-responder

Examples

- 2-tier: Messaging apps
- 3-tier: Web apps
- Event-driven: Ride sharing
- Peer-to-peer: Cryptocurrency
- Microservices: Social media

Combining patterns:

- **Can combine** 3-tier with microservices
- **Can be combine** Peer-to-peer with event-driven
- **Cannot combine** Peer-to-peer with 2-tier - in Peer-to-peer each node represents both client and server. But 2-tier architecture separates the client from the server.

Recap

In this video, you learned that:

- An architectural pattern is a repeatable solution to an architectural problem
- 2-tier: Has a client and server. Text messaging apps are an example
- 3-tier: Has 3-tiers that interact with each other. Web apps are an example
- Event-driven: An action is produced and responded to by a consumer. Ride-sharing apps are an example
- Peer-to-peer: Decentralized network of nodes that act as clients and servers. Cryptocurrency is an example
- Microservices: Loosely coupled individual services interact with the client and communication is orchestrated among services. Social media is an example
- Two or more patterns can be combined in a single system but some are mutually exclusive

Application Deployment Environments:

Things needed to run an application:

- Application code/executables
- Software stack like libraries, apps, middleware, OS
- Networking infrastructure
- Hardware like compute, memory, storage

Pre-production environment:



Production environment

- Entire solution stack ++
- Intended for all users
- Take load into consideration
- Other non-functional requirements
 - Security
 - Reliability
 - Scalability
- More complex than pre-production environments

Entire solution stack- both Hardware and Software

On-premises deployment:

- System and infra reside in-house within the organization's physical location often behind firewall. It prevents unauthorized access to / from private network
- If the organization desires great security on data and hardware it deploys the app in On-premises.
- Organization is responsible for system, Hardware and Software and maintenance to run the app

Types of cloud deployment models:

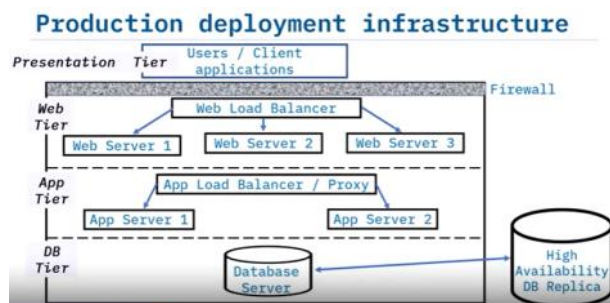


Recap

In this video, you learned that:

- Application environments include development, test/QA, staging, and production
- Production environments must also take into account non-functional requirements like load, security, reliability, and scalability
- Deployment platforms: on-premises, public, private cloud, & hybrid cloud

Production Deployment Components:



Firewall:

- Monitors traffic between an interior and an exterior network
- Permits or blocks data based on a set of security rules
- Acts as a barrier between networks to block viruses and hackers from accessing the internal network

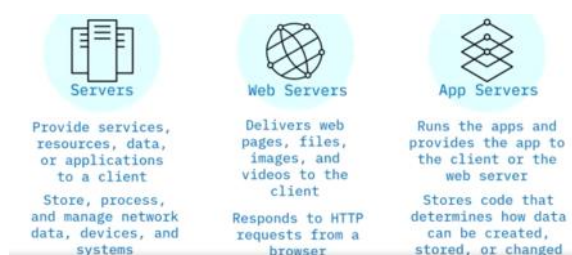
Load balancers:

Purpose: distributes traffic efficiently amongst multiple servers

Functions:

- Prevents server traffic overload
- Maximizes server capabilities and responsiveness
- Ensures no one server is overworked
- Manages concurrent requests fast and reliably

Web and Application Servers:



Proxy server:

Proxy server

- An intermediate server that handles requests between two tiers
- Can be used for load balancing, system optimization, caching, as a firewall, obscuring the source of a request, encrypting messages, scanning for malware, and more
- Can improve efficiency, privacy, and security

Database servers:

Database is a collection of related data stored on a computer that can be accessed in various ways. It controls the flow of data.

Recap

In this video, you learned that:

- Common production environment components include a firewall, a load balancer, web and application servers, proxy servers, and database servers
- A firewall is a security device that monitors traffic between networks
- Load balancers distribute network traffic among servers
- A web server delivers web content
- An app server provides business logic
- A database server stores and controls the flow of data through a DBMS

Job outlook for software engineer

27 September 2024 04:08 AM

Career path in software engineering:

Technical:

- Coding
- Problem solving
- Work with new technologies

Management :

Leadership qualities and soft skills

1. Public

- Software engineers should act to positively affect the public good

Accept responsibility with regards to:

- Safety
- Fairness
- Accessibility

2. Client/Employer

- Software engineers should act in the best interests of clients and employers

- Honest and forthright
- Seek consent when necessary
- Honor confidentiality

3. Product

- Software engineers should seek quality

Keep in mind:

- Cost
- Timelines

4. Judgement

- Software engineers should act with objectivity and honesty

- Maintain objectivity and honesty
- Do not engage in inappropriate financial activities
- Do not accept duties that create a conflict of interest

5. Management

- Managers and leaders should act in a manner consistent with these principles

- Minimize risk
- Employ security procedures
- Ensure realistic expectations
- Provide just compensation and intellectual property rights

6. Profession

- Software engineers should act to protect the reputation of the profession

- Comply with code of ethics
- Express concerns over violations

7. Colleagues

- Software engineers should treat colleagues with respect and fairness

- Encourage peers to comply with the code of ethics
- Do not take credit for work of others

8. Self

- Software engineers should be lifelong learners

- Endeavor to create quality software
- Strive to conduct themselves professionally

Terms used in Introduction to Software engineering

30 September 2024 04:22 AM

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 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

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

Requirement Gathering

01 October 2024 05:26 PM

User actions:

1. Self-Login
2. View paid amount details / total amount / due amount
3. View occasion details
4. Expected date to get the due amount
5. Occasions details, amount spent
6. All user payment details, balance amount

Admin actions:

1. Admin Login
2. Add new user
 - 1.1 user name
 - 1.2 date of birth
 - 1.3 joining date
 - 1.4 update paid money details - month, money,
 - 1.5 need to view each user payment history
3. Add Occasions - date, Name of people involved
 - 2.1 birthday celebration / marriage gifts
 - 2.2 Outings
 - 2.3 Team lunch
4. Add date, amount spent

System :

1. Calculate Total
2. Notification if not paid
3. Balance total amount - occasions spent
4. Birthday notification to all

Requirements

01 October 2024 10:05 PM

IKEA :

Software engineer:

- o Experience as a **Full Stack Developer** including **DevOps**
- o Proficiency in **NodeJS/Go** and **Python**
- o Proficiency in **Reactjs** and **Typescript**
- o Cloud native experience – **GCP preferred** using Cloud Run as compute, or experience with **atleast one** of the **cloud** technologies (AWS, Azure etc.) works
- o Strong knowledge of **GIT** and **Jenkins**
- o Hands-on experience with **Kubernetes**
- o Familiar with **Event Driven Architecture, Synchronous / Asynchronous APIs, OpenTelemetry**
- o **Collaborative** tools like **Git, Jira, Confluence**
- o **CI/CD** tools like **Github actions, containerisation** and **serverless systems** like **Docker, Kubernetes, etc.,**

WeekDay:

- o Proficiency in **Python with 2+ years** of experience.
- o Experience designing and writing **RESTful APIs**.
- o **Strong** knowledge of **AWS services** and **components**, particularly **serverless architectures (Lambda, API Gateway, etc.)**.
- o Familiarity with web frameworks such as **Django, Webapp2, Flask**, or similar.
- o Experience with **NoSQL** databases is a **plus**.
- o AWS Certification is preferred but not mandatory.
- o Familiarity with **Node.js** and **Google App Engine** is a bonus.
- o Educational background in Computer Science, Engineering, or a related field (B.E./B.Tech/M.Tech).
- o At least **2+ years of experience in backend/cloud development**.

Responsibilities:

- o Writing efficient, reusable, testable, scalable and clean code
- o Understanding, analyzing, and implementing the provided requirements
- o Guide and work with team members
- o Writing unit tests using python test framework tools
- o Troubleshooting on issues reported from Developers and proposing solution.
- o Relentlessly document your implementation, patterns, practices, and processes.
- o Bring in latest technologies and develop PoCs for problem statements.

Cloud computing

03 October 2024 04:25 AM

Why cloud computing:

- Flexibility & Scalability
- A business model with the latest trends
- Extra bandwidth and faster internet speed
- Customize the application cloud services from anywhere using internet connection
- Efficient marketing

About cloud computing by NIST [National Institute of Standards and Technology]:

Model for enabling convenient, on-demand network access to a shared pool of configurable computing resources.

Example for computing resources:

Networks, Servers, Storage, Applications, Services

Cloud computing have:

- 5 essential characteristics - On-demand Self-service, broad network access (allowing users to access cloud resources from various devices and platforms), resource pooling, rapid elasticity and measured service [automatically tracks and optimizes resource usage]
- 4 deployment models
- 3 service models

1. On-demand Self-service:

- Access every day except service outage and security breach
- Available without exceptions

Example:

- 24-hour ATM
- Vending machine

2. Broad network access:

- Computing resources can be accessed through the network
- Public cloud services can be accessed using internet connectivity and browser capabilities through any device (like desktops and laptops, tablets, ipads, etc.)
- Intranet is sufficient for accessing cloud services thus it is need not to reach out the entire world.

3. Resource pooling:

- Saves on cost when using shared model
- Computing resources serve multiple customers using multiple tenant model
- Cloud resources are dynamically assigned and reassigned on demand
- Regardless of physical location.

4. Rapid Elasticity

Resources - Increase / decrease, Scaled up(in)/down(out) vertically

5. Measured service

- Pay for what you use / reserve as you go
- Not applicable to email services, social media sites, and services on a trial basis
- Measured service also termed as utility model of billing like electricity bill have to pay after certain period.

Service model later.....

History and Evolution of Cloud Computing:

1950 Large-scale mainframes

- High-volume processing power was available by using time sharing (allows multiple users or processes to share a computer's resources simultaneously) / resource pooling techniques.
- Using dumb terminals its sole purpose was facilitating access to the mainframes and multiple users could access data storage layer and CPU power from any terminal.

1950 Virtual machines (OS)-

- Within single physical node and mainframes there is multiple VM's was possible. This allows multiple distinct computing environment.
- Each VM had Guest OS, Memory, CPU and hard drives and shared resources.

Virtualization became a **technological driver** and catalyst.

Hypervisor :

- It is a small software layer that enable multiple operating systems to share same physical computing resources. Separates the VM's logically assigning each slice of the underlying computing power, memory, and storage.
- Prevents the VM from interfering with each other.

- As technologies and hypervisors was improved, thus it could share and deliver resources reliably some companies decided to make the cloud's benefits accessible to users . These users didn't have abundance of physical servers to create their cloud computing infra.
- Users could order cloud resources from larger pool of available resources and pay on-demand / Utility computing model.
- Utility computing model became one of the key drivers behind CC launching. This allows to switch to a more cash-flow friendly OpEX model.

Tharani : chirikashikatoki
Shanmugapriya : aririkatorindojikanoshikifuka

Key considerations for cloud computing:

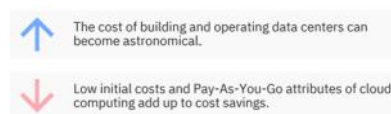
As company's transformation is unique for each company its cloud adoption strategy is also unique.

Key Drivers for moving to cloud



It is created without creating business disruption / issues related to security, compliance and performance.

1st consideration - Infra and work loads



Also, a point to consider is that not all workloads may be ready for the cloud, as is.

2nd consideration - SaaS and Development platforms

Paying for application access a more viable option / Purchasing off-the-shelf software and investing in upgrades.

3rd consideration - Speed and productivity

Cloud adoption - Flexibility (can scale back and scale up services, customize applications, access cloud service anywhere with an internet connection, determine level of control, pre-built tools, keep data secure using API keys), **Efficiency** (Get to market quickly without worrying infra costs and maintenance, Accessible from anywhere, Backed up on network) and **Strategic value** (provides innovative technologies, manages underlying infra)

Challenges of cloud adoption

- Data security associated with loss or unavailability of data
- Governance and sovereignty issues
- Legal, regulatory, and compliance issues
- Lack of standardization in evolving technologies
- Choosing the right deployment and service models
- Partnering with the right cloud service providers
- Business continuity and disaster recovery

Future of Cloud Computing



Alibaba Cloud



AWS



Google Cloud Platform

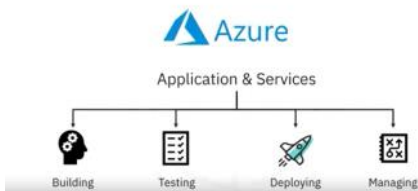


IBM Cloud



Also provides security, IOT and data analytics

Microsoft



Supports MS tools and frameworks, provides all IaaS, SaaS, PaaS

Oracle Cloud



Helping customers to personalize online and offline and mobile marketing campaigns for target audiences.

Salesforce - Focus on SaaS especially on customer relationship management.



Enterprise software and applications



Case studies using cloud technologies for:

- Better customer service
- Removal of barriers to innovation
- Achievement of enterprise scale
- Acceleration of growth

American Airlines

The Challenge:

- Improve customer experience and digital channels
- Improve response time to customer needs

The Solutions:

- New cloud-based technology platform
- Upgrade approach to delivering digital self-service tools
- Remove constraints of existing applications

The airline recognized the opportunity to remove the constraints of their existing customer-facing apps based on monolithic code (where all the components and functions of an application are tightly integrated into one unified codebase and user interface, business logic, and data access layers are combined into a single executable or project, with all components interacting directly) into cloud-native based microservices architecture on the cloud



Ubank:

Limit on head count - Restriction or cap on the number of employees or individuals that an organization or project is allowed to hire or retain

- Using PaaS
- More efficient ways to operate
- More control to developers
- Reduce the need for additional resources
- Faster speed to market
- Going from idea to production

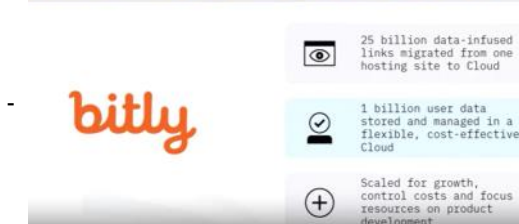
Removing Barriers to Innovation



Bitly:

- Shorten the URL for social media posts to an enterprise product.
- For cost-effective IT infra to support this transition it started planning for cloud migration.
- Needs are pay as you go, scale up and down, a global presence and the ability to geodistribute into more POPs at low risk.

Demand for Enterprise Scale



ActivTrades:

- Financial traders demand extreme speed and availability from trading systems.
- Enables investors to buy /sell on numerous financial markets.
- In needs if client base grew it wants to cut latency, accelerate execution and streamline the delivery of new functions.
- ActivTrades migrated 3 major trading systems from on-premises infra to IBM Cloud for Vmware solutions, backed by data storage, networking and security offerings on the IBM cloud.



Summary:

- Some case studies that demonstrate the impact businesses have created by adopting the cloud:
- American Airlines adopting cloud technologies to deliver customer value rapidly across its enterprise
- UBank leveraging cloud platform services to give more control to their developers, thereby removing barriers to innovation
- Bitly leveraging the scalability offered by cloud infrastructure for low-latency delivery to its geographically dispersed enterprise customers
- ActivTrades leveraging the infrastructure, storage, network, and security offerings on the cloud to accelerate execution and delivery of new functions in their online trading systems to their customers

IOT:

- Power, dynamic nature, scale and economics - cloud computing key enablers



These information are stored on the cloud

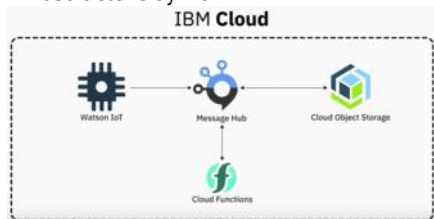
- IOT - AI(powers the insights), IOT(delivers the data), Cloud(AI and IOT leverage cloud scalability processing power to value to individuals and business alike) are like symbiotic relationship
- The collection point is near devices in motion

Welgevonden - To reduce the number of Rhino's were poached at combat sensor are set in zebra body to track rhino's activity.

Tennis stadium- IOT used to gather the players emotion, how they play (highlight plays to coach their players in best way) , brands used by the viewers (like t-shirt, foods)

Blockchain:

- Traceability in transactional applications
- A secure immutable network that allows members to view only the transactions that are relevant to them.
- Like IOT, block chain(provides the trusted and decentralized source of truth) have 3 way relationship with AI(Powers the analytics and decision making from the data collected) and cloud(provides globally distributed, scalable, and cost-efficient computing resources)
- Blockchain on the cloud helps farmers reduce waste by building traceability and transparency in the food supply chain
- The use of data analytics for driving predictive maintenance solutions for a city's infrastructure by KONE



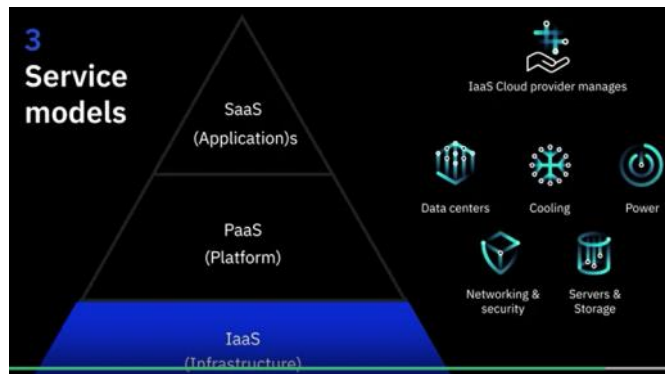
3 Service models:

IaaS - Infrastructure As A Service

- Set of compute, networking, storage resources that have been virtualized by vendor.
- IaaS is a set of compute networking and storage resources that have been virtualized by a vendor so that a user can access and configure them any way they want.
- In design we have a concept of talking about users called personas, Called system

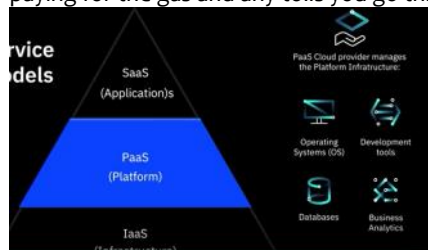
admin. Or an IT admin.

- IaaS is like **leasing a car**. So if you've ever leased a car, you probably did a lot of research, and you care about the specs of the car and the performance. You care about the color of the car, what kind of car it is. You're the one driving and you're paying for it. You're also paying for the gas and any tolls or maintenance



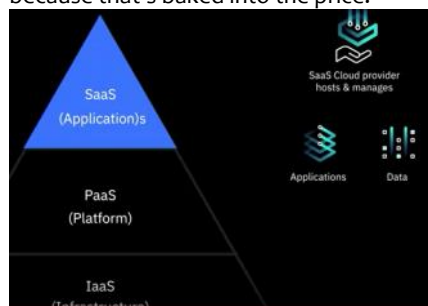
PaaS - Platform As A Service

- PaaS takes advantage of all the virtualized resources from IaaS and then just abstracts them so the user doesn't have to worry about managing any of those virtualized resources.
- The user for PaaS is usually a dev. In IBM we call this dev "Jane." is called persona
- Platform as a Service in this metaphor, more like renting a car. So say you're on vacation and you just got off the airport and you're going to pick up a rental car. You don't really care what color it is.
- You don't really even care about the specs of it, but you're still driving and you're paying for the gas and any tolls you go through.



SaaS - Software As A Service

- Software as a Service is just software that you don't have to install on your machine and you don't have to manually update.
- And so, the user for Software as a Service could be anyone.
Ex: YouTube - charged on a subscription model, rather than a one-time license fee
- Software as a Service is again the easiest one. That one's more like getting a taxi or an Uber. So with the taxi or an Uber, you don't care at all about what kind of car it is, what color it is and in fact, you're not even the one driving, or paying for gas, or any tolls because that's baked into the price.



IaaS:

- form of cloud computing that delivers fundamental compute, network, and storage resources to consumers on-demand, over the
- internet, on a pay-as-you-go basis.
- The cloud provider hosts the infrastructure components traditionally present in an on-premises data center as well as the virtualization
- or hypervisor layer.
- can create or provision virtual machines (or VMs) in their choice of Region and Zone

available from the Cloud Provider.

- pre-installed customer's choice of operating system
- The customers can then deploy middleware, install applications, and run workloads on these VMs. create storage for their workloads and backups.
- customers can track and monitor the performance and usage of their cloud services.

key components of cloud:

infrastructure:

Physical data centers:

- IaaS providers manage large data centers that contain the physical machines required to power the various layers of abstraction on top of them.
- In most IaaS models, end users do not interact directly with the physical infrastructure but experience it as a service provided to them.

various layers of abstraction:

- Physical infrastructure (physical hardware like servers, storage devices, and networking components)
- Virtualization Layer - allows multiple virtual machines (VMs) or containers to run on the same physical hardware.

Compute:

- IaaS providers manage the hypervisors and end-users programmatically provision virtual instances with desired amounts of compute, memory, and storage resources.
- comes with supporting auto scaling and load balancing

Network:

- Users get access through virtualization or programmatically, through APIs.

Storage:

There are three types of cloud data

- storage: object, file, and block storage.
- common mode of storage in the cloud, it is highly distributed and resilient.

IaaS supports a wide array of use cases:

- Organizations set up test and development environments faster, helping create new applications more quickly.
- By abstracting the low-level components, cloud infrastructure is helping developers focus more on business logic than infrastructure management.
- Business continuity and disaster recovery require a significant amount of technology and staff investments.
- IaaS is helping organizations reduce this cost and make applications and data accessible as usual during a disaster or outage.
- Organizations deploy their web applications faster and also scale infrastructure up and down as demand fluctuates.
- Organizations are leveraging cloud infrastructure to solve complex problems involving millions of variables and calculations such as climate and weather predictions and financial modeling.
- Mining massive data sets to locate valuable patterns, trends, and associations requires a huge amount of processing power.

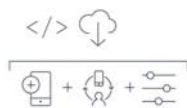
concerns :

- lack of transparency in the cloud infrastructure's configuration and management and dependency on a third-party for workload availability and performance

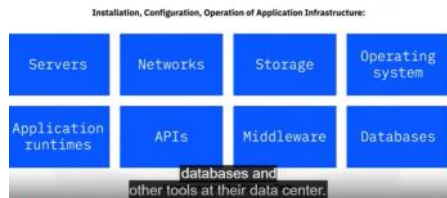
What is PaaS?

a cloud computing model that provides a complete application platform to

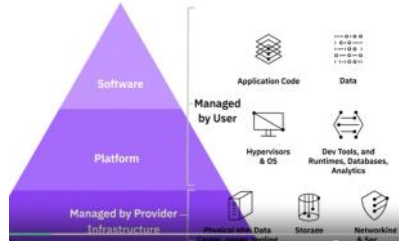
- Develop
- Deploy
- Run
- Manage



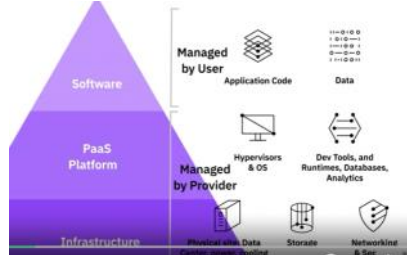
PaaS Providers Host & Manage:



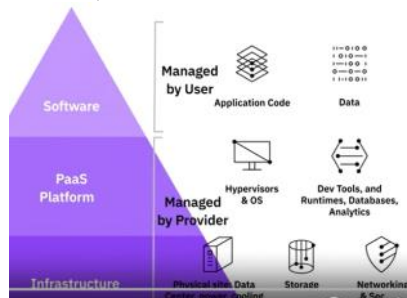
- The provider also takes responsibility for the installation, configuration, and operation of the application infrastructure, leaving the user responsible for only the application code and its maintenance.
- For **IaaS**,



- For **PaaS**,



- For **SaaS**,



high level of abstraction they provide to the users

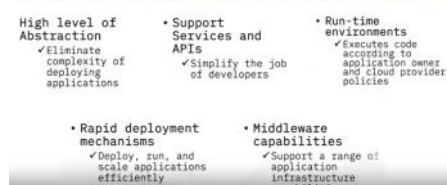
- eliminating the complexity of deploying applications, configuring infrastructure, and provisioning and configuring supporting technologies like load balancers and databases.

PaaS clouds provide services and APIs

- simplify the job of developers in delivering elastically scalable highly available cloud applications.
- These services typically include a variety of capabilities such as APIs for distributing, caching, queuing and messaging, file and data storage, workload management, user identity, and analytics, thus eliminating the need to integrate disparate components.

PaaS offerings support a range of application infrastructure or **middleware capabilities** such as application servers, database management systems, business analytics servers, mobile backend services, integration service business process management systems, rules engines, and complex event processing systems.

Essential Characteristics of PaaS



Use cases of PaaS:

API development and management.

- Organizations are using PaaS to develop, run, manage and secure APIs and microservices, which are loosely coupled, independently deployable components and services.

Internet of Things, or IoT:

PaaS clouds support a broad range of application environments, programming languages, and tools used for IoT deployments.

Business analytics, or intelligence:

PaaS tools allow organizations to analyze their data to find business insights that enable more informed business decisions and predictions.

Business Process management, or BPM.

Organizations are using the PaaS cloud to access BPM platform delivered as a service.

Master data management, or MDM:

Organizations are leveraging the PaaS cloud to provide a single point of reference for critical business data, such as information about customer transactions and analytical data to support decision-making.

Advantages of using PaaS:

Scalability - rapid allocation and deallocation of resources with a pay-as-you-use model offered by PaaS.

faster time-to-market - The APIs support services and middleware capabilities assist developers in focusing their efforts on application development and testing, resulting in faster time-to-market for their products and services.

Middleware capabilities reduce the amount of code that need to be written while expanding the application's functional capabilities.

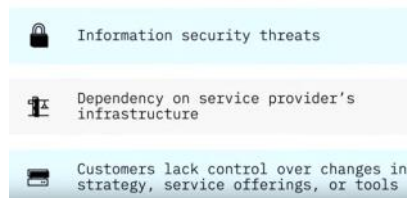
Greater agility and innovation- because using PaaS platforms means that you can experiment with multiple operating systems, languages and tools without having to invest in these resources. You can evaluate and prototype ideas with very low risk exposure, resulting in faster, easier, less risky adoption of a wider range of resources.

PaaS available offerings:

PaaS available offerings



Risks of PaaS



What is SaaS?

Software-as-a-service: a cloud offering that provides access to a service provider's cloud-based software.



and users use these applications without having to maintain and update the infrastructure.

Saas:

SaaS Supports



1. Multitenant architecture
2. Manage privileges and monitor data
3. Security, Compliance, Maintenance
4. Customization are limited -
5. Users pay via a subscription model
6. Scalable resources

Key benefits :

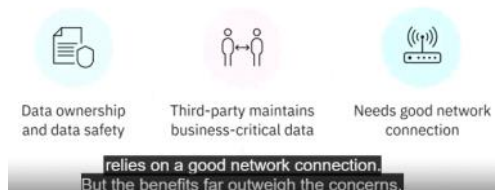
- Greatly reduce the time from decision to value
- Increase workforce productivity and efficiency
- Users can access core business apps from anywhere
- Buy and deploy apps in minutes
- Spread out software costs over time

Use cases:

Organizations are moving to SaaS to:

1. Reduce on-premises IT infrastructure and capital expenditure
2. Avoid ongoing upgrades, maintenance and patching
3. Run applications with minimal input for example email servers and office collaboration and productivity tools.
4. Manage websites, marketing, sales and operations
5. Gain resilience and business continuity of the cloud provider

Concerns:



4 Deployment models are :

1. Public cloud
2. Private cloud
3. Community cloud
4. Hybrid cloud

Public cloud:

In a public cloud users can access (these are resources):

- Servers
- Storage
- Network
- Security
- Applications

Using web consoles and API's users can provision the above resources services they need

- The cloud provider owns, manages, provisions and maintains the infrastructure
- Users don't own the servers their applications run on / storage their data consumes / manage the operations of the servers/ determine how the platform are maintained.
- Cost saving in terms of Total Cost for Ownership
- Subject to the performance and security of the cloud provider's infrastructure

Public cloud providers in the market today:

- AWS
- Azure
- IBM Cloud
- Google Cloud
- Alibaba Cloud

Everyone provides Servers, Storage, Network, Security and Databases with vary payment options

Public cloud characteristics:

Virtualized multi-tenant architecture enabling users to share computing resources residing out their firewalls.

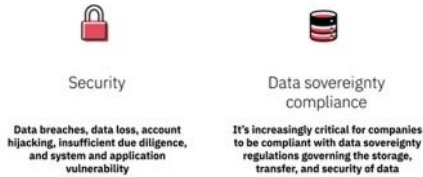
Resources are not dedicated for use by a single tenant / organization

Public cloud benefits:

1. Vast on-demand resources are available



Public cloud concerns:



Public cloud use cases:

Public cloud use cases



Private Cloud:

1. Exclusively used by a single organization with multiple consumers
2. Owned, managed and operated by organization, third party, or combination
3. May exist on or off premises
4. Could be implemented internally[Owned and managed by organization] and externally[Owned and managed by service provider]
5. A private cloud is a virtualized environment modeled to bring in the benefits of a public cloud platform without the perceived disadvantages of an open-end shared public platform.

Best of both worlds

Public Cloud Benefits:

- Dynamic scalability
- Cost efficiency
- Self-service

Control:

- Access
- Security
- Compliance

security a
compliances s

External Private cloud resides on a cloud service providers infrastructure is called Virtual Private Cloud (VPC) provided by AWS and IBM

Benefits of Private cloud:

1. The ability to leverage the value of cloud computing using systems that are directly managed or under perceived control of the organization's internal IT.
2. The ability to better utilize internal computing resources, such as the organization's existing investments in hardware and software, thereby reducing costs.
3. Better scalability through virtualization and cloud bursting["bursts" into a public cloud when the demand for computing resources spikes beyond the capacity of the private environment]. That is, leveraging public cloud instances for a period of time, but returning to the private cloud when the surge is meet.
4. Controlled access and greater security measures customized to specific organizational needs.
5. The ability to expand and provision things in a relatively short amount of time, providing

greater agility.

Common use cases:

1. Organizations may choose to opt for private cloud because of various reasons, because their applications provide a unique competitive advantage. There are security and regulatory concerns, or because their data is highly sensitive and subject to strict industry or governmental regulations.
2. Opportunity for organizations to modernize and unify their in-house and legacy applications. Moving these applications from hardware to the cloud and allows them to leverage the power of the compute resources and multiple services available on the cloud.
3. Organizations are integrating data and application services from their existing applications with public cloud services. move them without compromising security / compliance.
4. Ability to build applications anywhere and move them anywhere without having to compromise security and compliance in the process.
5. Full control over critical security and compliance issues from within the environment of their dedicated cloud

Hybrid cloud:

- It connects organization's on-premise private cloud and third-party public cloud
- Gives flexibility to choose the optimal cloud for each application / workload.
- Workloads move freely as needs change between 2 clouds , mission-critical applications , capacity requirements on private cloud infrastructure while deploying the less sensitive and more dynamic workloads on the public cloud
- Leverage public and private clouds with "cloud bursting" returns when surge is met.

The 3 tenets:

1. Interoperability : Public & private clouds understands each other's API, configuration, data formats, authentication & authorization
2. Scalability - Cloud bursting
3. Portability - Move applications & data between on-premise, cloud systems & cloud service providers

2 Types of Hybrid clouds:

1. Hybrid Monocloud [One cloud provider]
2. Hybrid Multicloud [Can be deployed on any public cloud infrastructure]
3. Composite Multicloud [Move single applications and data to any cloud systems & cloud service providers]

Benefits of Hybrid clouds:

1. Security and compliance
2. Scalability and resilience
3. Resource optimization
4. Cost-saving

Hybrid clouds are complex and challenging to deploy and maintain since they involve synchronization, redirection, latency, security, portability and interoperability & compatibility of policies applications and data and so on.

Use cases of Hybrid cloud:

1. Software as a service integration
2. Data & AI integration
3. Enhancing legacy apps
4. Vmware Migration
5. Leveraging public cloud services

Key concepts of to make up a hybrid cloud architecture:

1. **Connect / Interoperability** - Connects private and public using linux, kubernetes, Istio, multi clustered management tools, message queues, brokers etc.
2. **Modernization** - scaling
3. **Security** - taking advantage of using public cloud by maintaining security.

Community cloud:

Perimeter security model - Defending the boundary or perimeter of a network from external threats, assumption is that everything inside the network is trusted, designed to protect the network's "edge" or boundary, keeping unauthorized users and malicious entities out.

Data localization - Practice of requiring data to be stored, processed, and accessed within the geographic boundaries of a specific country or region.

Data sovereignty - Data is subject to the laws and regulations of the country where it is collected, processed, or stored.

Community -

- A Specific Company or Organization - single company or a group of companies that collaborate within the cloud while maintaining secure separation
- Role-Based Users or Departments: users within an organization who require access to specific shared resources based on their roles.
- An Industry or Sector: Banks collaborating to share best practices or secure data pools while complying with financial regulations like GDPR or PCI DSS.
- The community cloud members work under the same set of security controls.
- Provides the members the same attributes like citizenship and authorization controls while giving limited physical and/or logical access to resources.
- Supports data localization and some data sovereignty requirements based on the location of the community cloud's data centers.
- Perimeter security model encompassing the community cloud.

Implementation of software-defined community cloud :

To establish a security perimeter, Google Cloud is a software-defined approach that provides security and compliance assurances without the strict physical infrastructure constraints of legacy approaches. The Google community clouds use a combination of technologies referred to as “assured clouds” that can:

- Define communities around common projects, security and compliance requirements, and policy.
- Separate shared community projects from other projects.
- Modify capabilities of a community's boundary based on policy-controlled and audited configuration changes.

Software defined community cloud as a new type of “Government Cloud”:

In Google Cloud Platform (GCP), project is a unique, logical grouping of “infrastructure primitives.

atomic unit of capacity in GCP – a virtual machine (VM), a persistent disk (PD), a storage bucket, and others. Projects are “global resources” that can be assigned infrastructure primitives from any region or zone.

- Each project is separated from other customers' projects.
- Low-level resources like hypervisors, blocks in our distributed blockstore that underlies Google Cloud Storage (GCS) other components are isolated with resource abstractions
- Cloud infrastructure [that] is provisioned for exclusive use by a single organization — owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
- When a project is created within GCP, infrastructure primitives that are assigned to that project are scoped to only that project(creates "Enclave")
- When overlaid with Assured Workloads constraints for data residency, support personnel attributes, and security controls common to that community, these per-project private cloud enclaves become software-defined community clouds.

Infrastructure primitives :

Basic building blocks or foundational components of a cloud computing environment or IT infrastructure

Compute Primitives:

- **Virtual machines (VMs)** - physical servers
- **Containers** - portable units for running applications
- **Serverless functions** (e.g., AWS Lambda, Google Cloud Functions) - Units of compute , run code in response to events.

Storage Primitives:

Object storage - unstructured data like images or videos

Block storage - low-latency, high-performance data storage

File storage (e.g., AWS FSx, Google Filestore) - For shared files and directories

Network Primitives:

Virtual Private Networks (VPNs) - Securely connect different parts of an infrastructure

Load balancers - Distribute incoming traffic across servers.

Firewalls - Control traffic based on security rules.

Identity and Access Management (IAM) Primitives:

Users, roles, and permissions - Define who can access what resources

Authentication mechanisms - Like single sign-on (SSO) or multi-factor authentication (MFA)

Monitoring and Logging Primitives:

Metrics collection - Tools to monitor the performance of infrastructure.

Logging services - Capture logs from applications or services for debugging or auditing.



community
-cloud-1

After choosing cloud service model and cloud type, need to plan the infrastructure architecture.

Region:

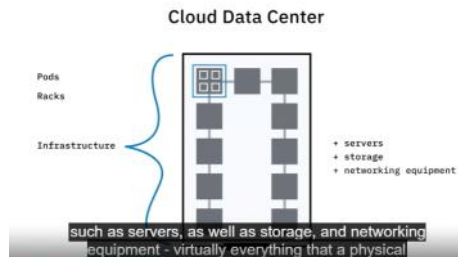
- Geographic location where infra is clustered may have names like NA south, US East
- Regions are isolated from each other does not impact natural disaster

Availability zones:

- Each region can have multiple availability zone (having own power, cooling, networking resources named like DAL-09/US-East-1)
- Improves fault tolerance, decreases latency and single shared point of failure
- Connects with other AZs and (data centers within AZs) connected to each other
AZs, regions private data centers, internet

Data center:

- Huge room/ warehouse containing cloud infrastructure.
- DC contains Pods and Racks



Computing resources:

- Compute options are - virtual servers, Bare metal servers and serverless computing resources
- Servers in DC run hypervisors to create virtual servers
- Users can run their workloads on serverless computing resources which has an abstraction layer on top of VM

Networking:

- Includes traditional networking hardware like - routers, switches
- By using SDN(Software Defined Networking) [here networking resources are virtualized / made available programmatically through APIs] allows easier networking includes provisioning, configuration, management.

Definitions

09 October 2024 05:22 PM

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

My doubts

10 October 2024 04:12 AM

QUES : As a customer I choose IaaS means how should I manage data center

ANS :

1. As a customer, the management of the data center infrastructure is largely abstracted, as the cloud service provider handles most of the physical and foundational aspects of the data center
2. significant responsibilities and control over your virtual infrastructure

What the Provider Manages in IaaS

- Physical servers, storage, and networking
- Data center facilities, cooling, power, and hardware maintenance
- Network security at the data center level
- Virtualization layer (e.g., hypervisors)

Applications (Customer)	<-- Your apps: databases, web servers,
Runtime Environment (Customer)	<-- Software frameworks, libraries, runtime.
Operating System (Customer)	<-- OS management: updates, patches, etc.
Virtual Machines and Storage (Customer/Provider)	<-- VM creation: disk sizes, scaling, etc.
Virtualization Layer (Provider)	<-- Abstracts physical hardware.
Physical Infrastructure (Provider)	<-- Physical servers, storage devices.
Networking (Provider)	<-- Physical and logical networking.
Data Center Facilities (Provider)	<-- Power, cooling, redundancy, etc.

VM - Create and configure VMs using provider APIs /portal. Scale up & scale out

Storage - Organize the data stored , manage regular backups

Network - Set up virtual networks (VPCs, subnets), Define rules to control access to and from your VMs. Establish secure connections between on-premises systems and your IaaS infrastructure.

HTML/HTML5

14 October 2024 04:09 AM

What is HTML?

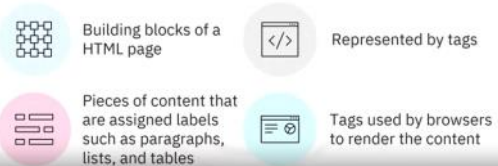


Hypertext Markup Language (HTML):

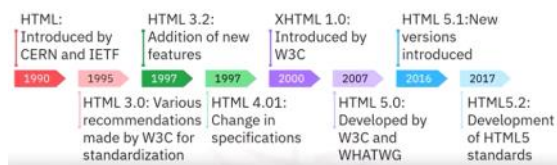
- language of the Internet
- Designed originally for sharing scientific documents
- Adapted to suit the requirement for displaying various types of documents as web pages

What is HTML?

HTML elements are:



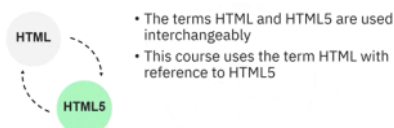
Evolution of HTML



Objectives of HTML5

- Defines a single language which can be written in HTML or XML syntax
- Interoperability with earlier HTML implementations
- Improves markup for documents
- Includes markup and APIs for idioms, such as web applications

HTML and HTML5



HTML example

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample Page</title>
  </head>
  <body>
    <p>Example paragraph</p>
    <!-- this is a comment -->
  </body>
</html>
```

- Declaration tag that represents the document type
- Instructs the web browser about the version of HTML
- Should be the first line of the HTML code

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample Page</title>
  </head>
  <body>
    <p>Example paragraph</p>
    <!-- this is a comment -->
  </body>
</html>
```

- Root element of the tree
- Contains all HTML elements except the <!doctype> tag

```

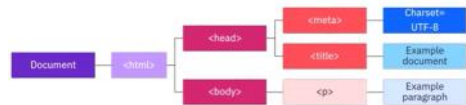
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample Page</title>
  </head>
  <body>
    <p>Example paragraph</p>
    <!-- this is a comment -->
  </body>
</html>

```

<head> element can contain the following tags:

- title (<title>)
- scripts (<script>)
- style (<style>)
- style sheet links (<link>)
- meta information (<meta>)
- browser support information and other initialization functions (<base>)

HTML DOM tree



```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="https://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type"
    content="application/xhtml+xml" charset="ISO-8859-1" />
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>

```

A document transmitted with an XML content type:

- Is treated as an XML document by a web browser
- An XML processor parses the document

Using HTML or XHTML

XHTML	HTML
<ul style="list-style-type: none"> • Tags need to be in lowercase • Codes must be well-formed • XML parser will stop processing if it encounters a situation where the syntax is not well-formed 	<ul style="list-style-type: none"> • Case used does not matter • Unmatched quotation marks, non-terminated and uncontained elements are allowed • Syntax is less rigorous than XHTML syntax

- HTML5 includes features for:
 - Categorizing sections of web pages
 - Managing data, video, and audio tools
 - Developing cross-browser and cross-platform applications
 - Creating engaging user experience

What is JSFiddle?

JSFiddle is a playground for novice and experienced web developers alike. It is an online web application which allows anyone to play around with HTML, CSS, and JavaScript. As you make changes, you can see the impact of those changes live.

Key themes of HTML5

- Syntax is compatible with HTML4 and XHTML1 documents
- Separates conformance requirements for user agents and authors



Supports elements and attributes of the earlier specifications for user agents



Removes several elements and attributes to simplify the language for authors

Key themes of HTML5

- HTML5 includes elements and APIs that help in creating web applications



Video and audio elements



API for drag and drop



API that supports offline web applications

HTML5 for web applications



What makes HTML suitable for creating web applications?

- Modern browsers support the full range of HTML5 features
- APIs enhance user experiences, such as advanced animation, drawings, audio, and video elements
- Use of HTML and CSS improves page load time
- SEO uses keywords provided by meta tags for Google and other search engines

```
<meta name="keywords" contents="coding, html production,css"/>
```

*Efficient use of HTML improves page load time

HTML5 elements

- Structural elements help to logically define the page structure
 - section, article, header, footer, figure, and figcaption
- APIs help with graphics and embedded content
 - canvas, audio, and video
- Input elements can automatically be validated by the browser
 - tel, email, datetime, number, range, and color
- Web storage APIs can store data in the browser
 - localStorage, sessionStorage
- Web workers run processing-intensive tasks without blocking the user interactions to the current page

The <head> tag defines the metadata about the page, which is used by browsers and search engines. This information may include the document title, character set, styles to be used, scripts, etc. Contents of <head> section are not displayed to the user.

TAG	DESCRIPTION
<mark>	Represents highlighted text
<meter>	Used for measurements with minimum and maximum values
<nav>	Specifies navigation for a document
<output>	Represents the result of a calculation
<progress>	Specifies the state of work in progress
<section>	Defines sections in a document or article
<source>	Used to specify multiple media resources for media elements
<summary>	Defines a visible heading for the details tag
<time>	Used to specify the date or time in a document
<video>	Specifies video, such as a movie clip or video stream
<!-- -->	Comments in source that are not displayed in the browser

Structural elements

- <article>
 - Represents a block of code that can logically stand alone
 - Examples include a newspaper article, user comments, blog entries
 - Used with the <time pubdate ...> attribute to provide a publication date for the article
- <section>
 - Represents a logical separation in a document, for example, chapters of a book
 - Accompanied with a heading

Example with structural elements

```
<body>
<article>
  <header>
    <h1>GB summit protests</h1>
  </header>
  <div>
    <section id="public">
      <h2>Public demonstrations</h2>
      <p>...more...</p>
    </section>
    <section id="control">
      <h2>Crowd control</h2>
      <p>...more...</p>
    </section>
  </div>
  <footer>
    <p>Published <time datetime="2018-12-20">Dec 20, 2018</time></p>
  </footer>
</article>
</body>
```

- <aside>
 - Provides additional information that is related to the main discussion
 - Extracts and displays further content or navigates to additional resources without detracting from the main discussion
 - Used for cautions, notes, or to wrap navigational links
- <figure>
 - A self-contained element referred to from the main content
 - Embeds graphics, photographs, or code segments
 - Can be moved to an appendix without affecting the flow of the document
- <figcaption>
 - Defines the caption for the contents of the <figure> element

- Navigational links are placed inside a <nav> tag

```
<body>
<nav>
  <div class="menu">
    <a href="#">Home</a> |
    <a href="about.html">About</a> |
    <a href="register.html">Register</a> |
    <a href="#">Sign in</a>
  </div>
</nav>
<h1>Pages for you</h1>
</body>
```

Div tag:

- <div> tags are commonly used when many elements are required to have the same format.
- Grouping such elements together in the same <div> tag enables a developer to easily style them by either using a class or an id.
- When using a <div> tag, note that browsers will insert a line break before and after the element.

Section tag:

- content in a more specific way than the <div> tag, Content within a <section> tag has a theme, which is usually indicated by a heading tag (e.g. <h1>) used immediately after the opening <section> tag.

```
<section>
  <h1>Section 1</h1>
  <p>This is text related to section 1.</p>
</section>

-

<section>
  <h1>Section 2</h1>
  <p>This is some text related to section 2.</p>
</section>
```

G8 summit protests

Public demonstrations

...more...

Crowd control

...more...

Published today.

Article tag:

- used to group together semantically related, self-contained content which can be meaningful on its own. Similar to the <section> element, articles usually have headings immediately after their opening tag to indicate what the article is about.

```
<article>
  <h1>Article 1</h1>
  <p>This paragraph is related only to article one and is meaningful on its own, without the rest of the code.</p>
</article>

-

<article>
  <h1>Article 2</h1>
  <p>This paragraph is related only to article two and is meaningful on its own, without the rest of the code.</p>
</article>
```

Aside tag:

- used to provide additional information related to the main discussion.
- It lets you display further content or additional resources without detracting from the main discussion, and is often placed as a sidebar in a document.
- While the <aside> element itself is not displayed different from the rest of the content, it is useful for understanding your code and for styling purposes.

```
<p>This is a paragraph of text that casually mentions a concept and continues...</p>

<aside>
  <h4>Concept Definition</h4>
  <p>This goes over the concept mentioned in the paragraph to provide more detail...</p>
</aside>
```

Figure and figcaption tag:

- self-contained content, such as a diagram or photo, that is referred to from the main content. The content within the <figure> element should be related to the main content, but still independent in such a way that if removed from the document, it would not affect the flow.
- The <figcaption> tag defines the caption for the contents of the <figure> element. It can be placed as the first or last child element within the <figure> element.

```
<figure>
  
  <figcaption>IBM Developer Skills Network Logo</figcaption>
</figure>
```

Nav tag:

- The <nav> tag is a convenience tag which does not alter the appearance on webpages. However, it is useful in styling all navigational elements, as well as omitting the content for certain functionalities, such as with screen readers.

Audio tag:

- used to embed sound content, such as music or podcasts. It contains one or more <source> tags with different audio sources (e.g. MP3, WAV), so the browser can select the first supported source to play.
- If a browser does not support the audio formats provided, the browser will instead display any text within the <audio> element.

```
<audio>
  <source src="soundtrack.mp3" type="audio/mpeg">
  <source src="soundtrack.ogg" type="audio/ogg">
  Your browser does not support the audio formats provided.
</audio>
```

Embed tag:

- used as a container to embed external resources such as media players and plug-in applications into your web page.
- <embed type="text/html" src="another_webpage.html">

Track tag:

- The <track> element defines text tracks, such as subtitles or captions, for <audio> and <video> elements.
- This text should be visible as the related media source it playing, and are formatted in the WebVTT (.vtt) format.

```
<video>
  <source src="common_html_elements.mp4" type="video/mp4">
  <track src="english_subtitles.vtt" kind="subtitles" srclang="en" label="English">
  <track src="spanish_subtitles.vtt" kind="subtitles" srclang="es" label="Spanish">
</video>
```

- The HTML <fieldset> tag is found within the <form> tag and is used to group related elements in an HTML form, often by enclosing them within a box.

disabled: It specifies that the elements belonging to the fieldset should be disabled.

form: It specifies the id of the form that the fieldset is to be considered a part of.

name: It specifies the name for the fieldset.

- A fieldset can additionally have a title or name, which can be provided by legend.

The <legend> tag is used with the <fieldset> element as a first child (the first inner tag) to define the caption for the grouped related fields.

```
<form>
  <fieldset name="personal_details">
    <legend>Personal Details</legend>

    <label for="fname">First name:</label>
    <input type="text" id="fname" name="fname"><br>
    <label for="lname">Last name:</label>
    <input type="text" id="lname" name="lname"><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email"><br>
    <label for="phone">Phone Number:</label>
    <input type="tel" id="phone" name="phone"><br>
  </fieldset>
</form>
```

Personal Details

First name:

Last name:

Email:

Phone Number:

Applying CSS to HTML

- Can use a combination of all three methods
- The type with the highest priority is applied

Highest
Priority

Inline > Internal > External

Lowest
Priority

Types of CSS frameworks

Plain/Vanilla CSS	Utility-first framework	Component framework
Developers write style for all components	Provides utility classes	Provides pre-styled components
Provides complete freedom when styling	Simplifies references to CSS properties	Requires little knowledge of CSS
Requires time and effort	Provides freedom when styling components	Assists in keeping consistent styles
		Limits freedom when styling

To change a link turn into red color while hovering on it

Vanilla CSS	Tailwind CSS
<pre>a { color: red; text-decoration: underline; } a:hover { color: rgb(185, 28, 28); }</pre>	<pre>Dangerous Link</pre>

Utility-first frameworks

Advantages	Disadvantages
Provides an easier way to reference CSS properties	Mixed styles reduces separations of concern, making HTML markup verbose
Utility classes scope to single-purpose CSS classes	Involves adding several classes to HTML markup
Implements CSS properties from HTML class attributes	Increases HTML download size and slows down web pages
Helps in consistency with color choices, spacing, typography, and shadows	

```
text-align: center; //CSS property
text-center //Utility class
```

Recap

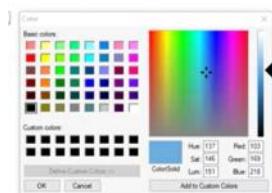
- CSS frameworks assist in implementing UI elements and creating dynamic web pages
- Plain (Vanilla) CSS lets developers write the styles and layouts of a website
- Utility-first frameworks provide utility classes to help build one's own styles and layouts
 - For example: Tailwind CSS
- Component frameworks provide pre-styled components and templates that can be implemented onto a website
 - For example: Bootstrap

- You specify a color for an element using a predefined color name, or by using RGB, HEX, HSL, RGBA, or HSLA values.

Common attributes of the input tag:

1. Number:

Input for color chooser: `<input type="color"/>`



- Allows a user to select color
- Dialog varies depending on browser
- Not all browsers support such input type
- Some browsers display the input type as regular text

Google Chrome

Date:

Ex: `<input type="date"/>`

1. A date control without timezone

Datetime :

Ex: `<input type="datetime-local"/>`

1. Provides for a date and time year,month,day,hour,minute, AM/PM without timezone.

Email:

Ex: `<input type="Email"/>`

Accepts only valid email format

Number:

Ex: `<input type="number"/>`

Takes numeric value as input

Range:

Ex: `<input type="range"/>`

Takes a numeric range as input


```
<input type="range" min="5" max="10">
```



Search:

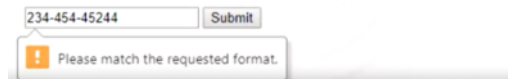
Similar to the text field, return a history of recently searched text strings.

Telephone :

Input of phone numbers: `<input type="tel">`

- Used for telephone numbers
- Provides a text entry field in the browsers
- Does not enforce numeric only input
 - Accepts characters, such as the plus sign and hyphens

```
<input type="tel" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}">
```



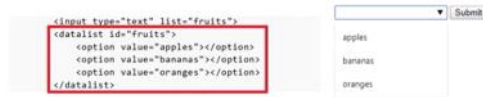
URL :

Validate and allows only properly formatted URL

Dataset:

Input attribute "list"

- Used with `<datalist>` tag that include the options list
- Useful for auto-complete functionality



Placeholder:

Input attribute "placeholder"

- Provides an example of the input text format in a lighter shade of text

```
<input type="email" placeholder="example@email.com">
```



- Does not send the placeholder text as input during form submission

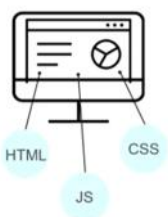
Required:

Indicates that the field is mandatory

Ex: `<input type="email" required>`

CSS Styling:

Media independence with HTML5, JavaScript and CSS



- Gives developers flexibility to target web and mobile devices
- Provides features that show interactive applications without requiring plug-in technologies
- Provides assistive technologies for vision impaired people
- Meets accessibility requirements over browser plug-ins for delivering interactive content

CSS design and coding

- Important concept to separate data from design:
 - Data is sent to the browser using HTML, and design is applied to that data using a CSS
 - Allows rendering of Web pages without design using special accessibility needs
 - Allows machines to index a Website without design interference
- CSS can be coded as a style attribute in an HTML tag, a head section of a document, or an external document
 - Preference is to code CSS in external documents

Style rules for the following web page elements:

1. Fonts
2. Text
3. Colors
4. Backgrounds
5. Sizes
6. Borders
7. Spacing
8. Positioning
9. Visual effects
10. Tables
11. Lists

CSS Format:

```
html-tag-name
{
  css-property-key-1: css-value-1;
  css-property-key-2: css-value-2;
}
```

HTML Elements	Description
Tags	<ul style="list-style-type: none">• Any tags in the HTML code• For example: <a>, <div>, , or <label>
ID reference	<ul style="list-style-type: none">• Displayed with a preceding hash symbol (#)• For example: #id-of-html-tag
Class reference	<ul style="list-style-type: none">• Displayed with a preceding dot/period symbol (.)• For example: .class-of-html-tag

Base style:

Body

```
{
background-color: #EEEEEE;
color: #000000;
margin: 0;
padding: 0;
text-align: left;
font-size: 100%;
font-family: sans-serif;
}
```

Guidelines for base styles:

- Colors use Red-Green-Blue (RGB) hexadecimal light values
- Size use pixels, em, or a percentage
- Text can be aligned left, right, or center
- Floats can also be left or right
- Vertical alignments must be top, middle, or bottom
- Fonts can be any specific font or font family, such as serif, sans-serif, or monospace or even a downloadable font

Choosing Layout:

Fluid Layout:

In a fluid layout:

- The height and width of elements are flexible
- The expansion or contraction is based on screen size
- The elements are specified using percentages and ems

Java script:

- Derived from the ECMA script standard
- Designed to run on netscape navigator browser
- Interpreted language
- Adds dynamic web contents

Wrapper objects:

I am kas mora

13 November 2024 05:14 PM

What should I ask for them