

## 2. Software Requirement Analysis & Specification

### 2.1 Product Perspective

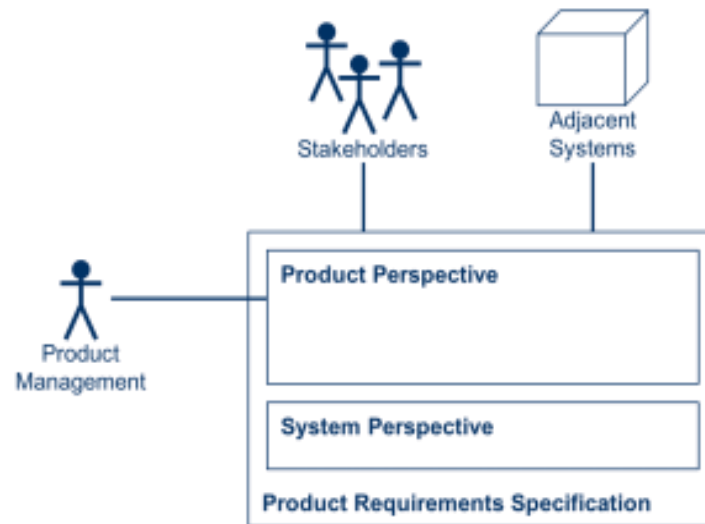
An essential aspect necessitates assurance, which involves bridging the gap between product management and development. It entails thoroughly defining a product in terms of its associated requirements, encompassing all the necessary specifications that must be explicitly described and continuously available.



**Fig.no: 2.1 Product Perspective**

### 2.2 Product Function

- In product management, the product requirements specification serves as the central tool, fulfilling the following purposes.
- It articulates how product management addresses partner needs and requests.
- It communicates to development teams what the product or new features to be developed will entail.
- The accompanying figure illustrates these relationships, focusing on the product perspective.
- Product management is responsible for defining a product perspective that aligns with the expectations and requirements of the product's stakeholders. Additionally, it should consider and serve relevant relationships with adjacent systems.



**Fig.no: 2.2 product perspective**

The key characteristics of the product perspective include.

- Describes the external view of the product.
- Answers the question "What".
- Represents what the product is.
- Describes what the product does.
- Addresses both the business and usage aspects of the product.
- Utilizes problem domain language and concepts.
- Presents a "Black Box" view, focusing on the external behavior and functionality of the product without delving into its internal workings.

## 2.3 User Characteristics

This section outlines the characteristics of users in three distinct contexts: the end user who interacts with the machine translation system; the end user of the final product of the translation process, which may include post-editing; the organization producing the machine translation system.

Relevant stakeholders may include.

- Interpreters
- Post-editors
- Translation buyers
- Translation managers
- Higher-level

## 2.4 Modules

- **Core Algorithm Implementation:** Development of the key generation algorithm for your new digital signature primitive. Implementation of the signing algorithm using the private key. Implementation of the signature verification algorithm using the public key.
- **Security Analysis and Proofs:** Formal security model definition for your new primitive. Mathematical proofs demonstrating the security properties of your scheme (e.g., unforgeability under chosen message attacks). Analysis of potential attack vectors and how your primitive mitigates them.
- **Performance Evaluation:** Benchmarking the key generation, signing, and verification speeds of your new primitive. Comparing its performance against existing digital signature schemes. Analyzing its resource consumption (e.g., computational power, memory usage, signature size).
- **Parameter Selection and Optimization:** Determining appropriate security parameters for your primitive. Exploring potential optimizations for efficiency and practicality.
- **Digital Signature Integration:** Developing mechanisms to use your new digital signature primitive for signing transactions within the blockchain. Implementing verification of these new signatures by blockchain nodes. Potentially modifying transaction structures or consensus protocols to accommodate your new primitive.
- **Smart Contract Interaction:** If your application involves smart contracts, you'll need modules to deploy and interact with them. Developing smart contracts that utilize your new digital signature scheme for authentication or authorization.

## 2.5 Functional and Non-Functional Requirements

### 2.5.1 Functional Requirements

Functional requirements delineate the intended actions of the system and can be segmented into various categories.

- **Inputs:** Specifies the data the system should accept.
- **Outputs:** Defines the results or data the system should generate.
- **Data Storage:** Determines the data that must be stored by the system.

#### **Input Design.**

Input design facilitates interaction between the information system and users, establishing processes for data preparation and entry. It involves methods to transform transaction data into usable formats, such as reading from documents or direct entry. Input design aims to streamline processes, minimize errors, and ensure user security and privacy. Key considerations include.

- Defining input data requirements.
- Organizing and coding data.
- Providing user guidance.
- Implementing input validations and error handling.

## 2.5.2 Non-functional Requirement

Non-functional requirements (NFRs) address fundamental aspects of software systems, focusing on qualities beyond specific functionalities. Failure to address NFRs adequately can result in user dissatisfaction, software conflicts, and increased time and cost to rectify issues.

### Types of Non-functional Requirements

- **Scalability:** The system's ability to handle increasing loads without compromising performance.
- **Reliability:** The system's ability to perform consistently and predictably under various conditions.
- **Regulatory Compliance:** Ensuring the system adheres to relevant regulations or standards.
- **Maintainability:** Ease of maintaining and updating the system over time.
- **Serviceability:** The ease with which the system can be repaired or serviced.
- **Utility:** The usefulness and value provided by the system to users.
- **Availability:** Ensuring the system is accessible and operational when needed.
- **Usability:** The system's ease of use and user experience.
- **Interoperability:** The system's ability to interact and operate with other systems or components.
- **Environmental Considerations:** Factors related to the environmental impact of the system, such as energy consumption or sustainability.

Addressing non-functional requirements is crucial for ensuring the overall success and effectiveness of a software system. Each requirement category contributes to the system's overall performance, reliability, and user satisfaction, making them essential considerations throughout the development lifecycle.

## **2.6 System Specifications**

### **2.6.1 Software Requirements**

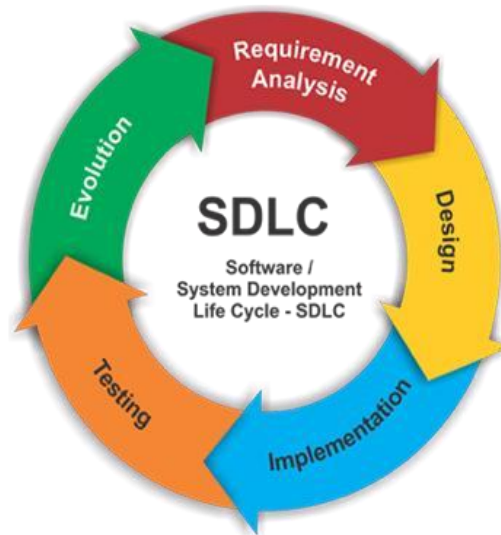
- Operating System : Windows XP
- Platform : PYTHON TECHNOLOGY
- Tool : Python 3.6
- Front End : Python anaconda script
- Back End : Spyder

### **2.6.2 Hardware Requirements**

- Keyboard : Standard Keyboard
- System : Pentium IV 2.4 GHz
- Hard Disk : 40 GB
- Monitor : 15 inch VGA Color
- Mouse : Logitech Mouse
- Ram : 512 MB
- Keyboard : Standard Keyboard

## 2.7 Software Development Life Cycles

Software Development Life Cycle Models and Methodologies



**Fig.no: 2.7 SDLC**

The Software Development Life Cycle (SDLC) is a series of stages that provide a structured approach to the software development process. It encompasses understanding the business requirements, eliciting needs, converting concepts into functionalities and features, and ultimately delivering a product that meets business needs. A proficient software developer should possess adequate knowledge to select the appropriate SDLC model based on project context and business requirements.

Therefore, it is essential to select the right SDLC model tailored to the specific concerns and requirements of the project to ensure its success. To explore more about choosing the right SDLC model, you can follow this link for additional information. Furthermore, to delve deeper into software lifecycle testing and SDLC stages, follow the highlighted links here

The exploration will cover various types of SDLC models, their benefits, disadvantages, and when to use them. SDLC models can be viewed as tools to enhance product delivery. Therefore, understanding each model, its advantages, disadvantages, and the appropriate usage is crucial to determine which one suits the project context.

Types of Software developing life cycles (SDLC)

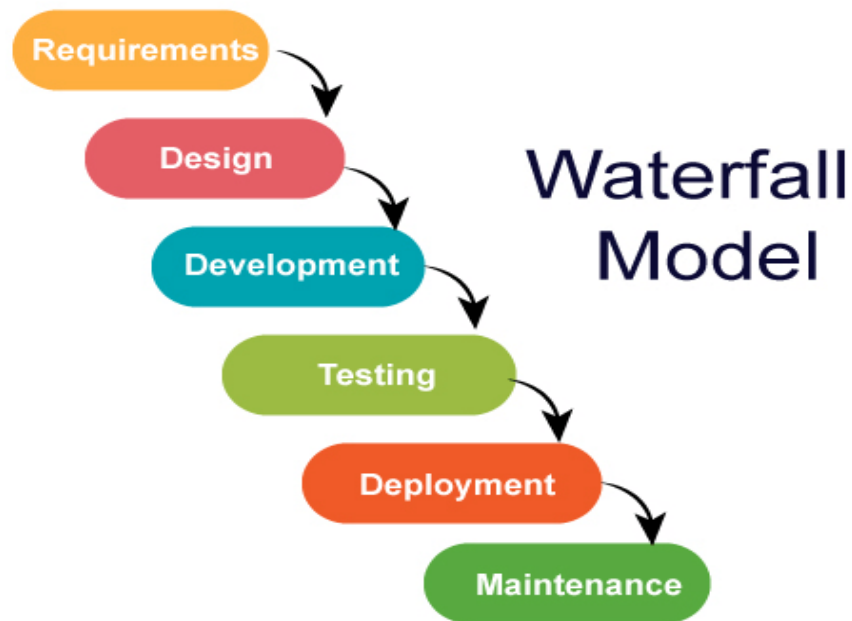
- Waterfall Model
- V-Shaped Model
- Evolutionary Prototyping Model
- Spiral Method (SDM)
- Iterative and Incremental Method
- Agile development

### **2.7.1 Waterfall Model**

The Waterfall Model follows a linear, sequential flow, where progress moves steadily downwards (like a waterfall) through the phases of software development. Each stage in the development cycle begins only after the previous stage is completed. The waterfall approach does not accommodate going back to a previous stage to address changes in requirements. It is the oldest and most well-known method used for software development. The five-stage waterfall model, based on Winston W. Royce's requirements, divides development processes into the following stages.

- 1.analysis
- 2.design
- 3.implementation
- 4.testing
- 5.operation

The waterfall model can be broken down into multiple phases



**Fig.no: 2.7.1 water fall model**

### **Advantages**

- Simple to clarify for the clients.
- Structures approach.
- Stages and exercises are distinct.
- Assists with arranging and timetable the task.

## **2.8 System Study**

The feasibility study evaluates the practicality of the project and enhances fundamental understanding through a comprehensive approach. During system assessment, the integrity evaluation of the proposed structure is crucial to ensure it does not burden the organization. Three key assessments conducted during feasibility appraisal are.

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

### **2.8.1 Economic Feasibility**

This study examines the financial impact of the system on the organization. It assesses the resources available for system development and justifies expenses. The design should be economically viable, leveraging freely available enhancements wherever possible, with consideration given to necessary purchases.

### **2.8.2 Technical Feasibility**

This study assesses the specific requirements of the system and ensures it does not overly strain existing technical resources. Excessive demands on technical resources can lead to burdens on users. The design should have modest technical requirements, minimizing unnecessary changes.

### **2.8.3 Social Feasibility**

This aspect examines the level of acceptance of the system by users. It involves establishing effective means to familiarize users with the system and ensuring they perceive it as a necessity rather than a threat. User confidence should be bolstered through clear communication and user training. Social feasibility encompasses analyzing how individuals interact within the system or organization and evaluating social impacts to understand their extent and scale.

## 2.9 Methodology and Algorithms

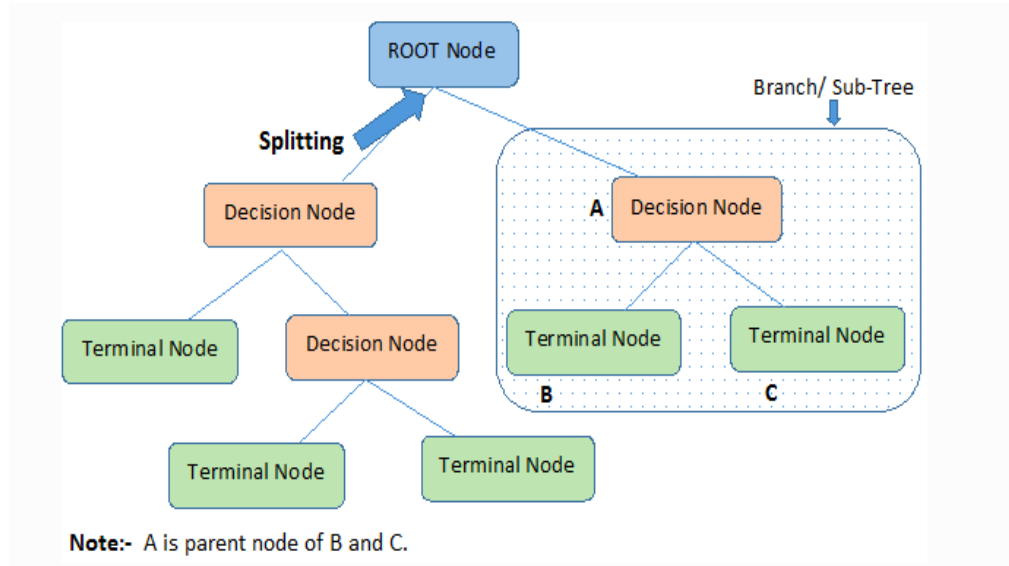
### Decision tree classifiers

Decision tree classifiers are widely applied across various domains due to their ability to capture descriptive decision-making knowledge from provided data. One of their most notable features is their capability to be generated from training sets. The process for generating decision trees based on a set of objects ( $S$ ), each belonging to one of the classes  $C_1, C_2, \dots, C_k$ , unfolds as follows.

**Step 1:** If all objects in  $S$  belong to the same class (e.g.,  $C_i$ ), the decision tree for  $S$  comprises a leaf labeled with this class.

**Step 2:** Otherwise, select a test ( $T$ ) with potential outcomes ( $O_1, O_2, \dots, O_n$ ). Each object in  $S$  has a specific outcome for test  $T$ , thus partitioning  $S$  into subsets ( $S_1, S_2, \dots, S_n$ ), where each object in  $S_i$  corresponds to outcome  $O_i$  for test  $T$ . Test  $T$  becomes the root of the decision tree, and for each outcome  $O_i$ , a subsidiary decision tree is constructed by recursively invoking the same procedure on set  $S_i$ .

### Block Diagram for Decision Tree Algorithm



- **Root Node**

It represents the entire population or sample and this further gets divided into two or more homogeneous sets.

- **Splitting**

It is a process of dividing a node into two or more sub-nodes.

- **Decision Node**

When a sub-node splits into further sub-nodes, then it is called the decision node.

- **Leaf / Terminal Node**

Nodes do not split is called Leaf or Terminal node.

- **Pruning**

When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.

- **Branch / Sub-Tree**

A subsection of the entire tree is called branch or sub-tree.

### **Gradient boosting**

Gradient boosting is a versatile machine learning technique employed in regression and classification tasks, among others. It constructs a prediction model in the form of an ensemble of weak prediction models, typically decision trees. When using decision trees as the weak learner, the resulting algorithm is referred to as gradient-boosted trees, often surpassing the performance of random forests. The construction of a gradient-boosted trees model occurs in a stage-wise manner, similar to other boosting methods, but it stands out by allowing the optimization of an arbitrary differentiable loss function.

**K-Nearest Neighbors (KNN)**

K-Nearest Neighbors (KNN) is a simple and intuitive machine learning algorithm used for both classification and regression tasks. The core idea behind KNN is based on the concept of "neighbors" — to make a prediction for a new data point, the algorithm identifies the 'K' closest data points in the training set, typically using a distance metric like Euclidean distance. For classification, the new data point is assigned the most frequent class among its nearest neighbours, while for regression, the prediction is usually the average of the neighbouring values. KNN is a non-parametric, instance-based learning method, meaning it makes decisions based on the entire training dataset without making explicit assumptions about the data distribution. Its simplicity and effectiveness make it a popular choice for many practical applications, especially when the dataset is small and the decision boundary is not overly complex.

**Logistic regression**

Logistic regression analysis explores the relationship between a categorical dependent variable and a set of independent variables. The term "logistic regression" is applied when the dependent variable has only two values, such as 0 and 1, or Yes and No. On the other hand, "multinomial logistic regression" is used when the dependent variable has three or more unique values, like Married, Single, Divorced, or Widowed. While the nature of data for the dependent variable differs from that of multiple regressions, the practical application of the procedure remains similar.

Logistic regression serves as a competitor to discriminant analysis in analyzing categorical-response variables. Many statisticians favor logistic regression due to its versatility and suitability for modeling various situations compared to discriminant analysis. This preference arises from logistic regression's ability to not assume that the independent variables follow a normal distribution, unlike discriminant analysis.

**Naive Bayes**

The naive Bayes approach is a supervised learning method founded on a simple assumption: it presumes that the presence or absence of one feature of a class is independent of the presence or absence of any other feature. Despite its simplicity, it demonstrates robustness and efficiency comparable to other supervised learning techniques. One explanation often highlighted in the literature is based on representation bias.

The naive Bayes classifier operates as a linear classifier, akin to linear discriminant analysis, logistic regression, or linear support vector machines (SVMs). However, the distinction lies in the method used to estimate the classifier's parameters, known as the learning bias. Although the naive Bayes classifier finds extensive use in the research community due to ease of programming, parameter estimation simplicity, rapid learning even with large datasets, and reasonably good accuracy compared to other methods, it remains less popular among practitioners seeking practical results.

**Random forests**

Random forests, also known as random decision forests, represent an ensemble learning technique used for classification, regression, and other tasks. They function by constructing numerous decision trees during training. For classification tasks, the output of the random forest is determined by the class selected by the majority of trees. Conversely, for regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests aim to mitigate the issue of decision trees overfitting to their training set.

The concept of random decision forests was first introduced in 1995 by Tin Kam Ho, who utilized the random subspace method. This method, as formulated by Ho, serves as an implementation of the "stochastic discrimination" approach to classification initially proposed by Eugene Kleinberg.

**Support Vector Machine (SVM)**

Support Vector Machine (SVM) represents a discriminant machine learning technique commonly used in classification tasks. It aims to find a discriminant function, based on an independently and identically distributed training dataset, that accurately predicts labels for newly acquired instances. Unlike generative machine learning approaches, which necessitate computations of conditional probability distributions, a discriminant classification function assigns a data point  $x$  to one of the classes involved in the classification task. Compared to generative approaches, discriminant methods may be less powerful, particularly in outlier detection scenarios. However, they require fewer computational resources and less training data, especially in multidimensional feature spaces and when only posterior probabilities are necessary. Geometrically, learning a classifier equates to identifying the equation for a multidimensional surface that optimally separates the different classes in the feature space.

SVM is a discriminant technique that solves convex optimization problems analytically, consistently yielding the same optimal hyperplane parameters. In contrast, genetic algorithms (GAs) and perceptrons, both widely used for classification in machine learning, may produce solutions highly dependent on initialization and termination criteria. With a specific kernel transforming data from the input space to the feature space, SVM training returns uniquely defined model parameters for a given training set, whereas perceptron and GA classifier models vary with each training iteration.

## **2.10 Technologies Used**

### **Python**

Python served as the core programming language for developing the software defect prediction model. Its simplicity and flexibility allowed for quick implementation and debugging of code. Python's extensive ecosystem of libraries such as NumPy, Pandas, Scikit-learn, TensorFlow, and Matplotlib made it easier to handle data manipulation, model training, and evaluation. Its object-oriented and functional programming capabilities supported complex algorithm development. Python's compatibility with machine learning frameworks helped in seamless model deployment. It also provided built-in support for HTTP requests, file handling, and data storage. The availability of pre-trained models and extensive documentation reduced development time. Python's open-source nature ensured cost-effectiveness. Its platform independence allowed smooth execution across different operating systems.

### **Django**

Django acted as the backend framework for the project, providing a structured way to develop the web application. Its Model-View-Controller (MVC) architecture separated logic, presentation, and data management. Django's built-in authentication system simplified user registration and login handling. The ORM (Object-Relational Mapping) feature allowed interaction with the database using Python code instead of SQL. Django's middleware supported secure handling of HTTP requests and responses. Its scalability allowed the system to handle increased loads effectively. Django's modular nature enabled easy integration of external libraries. Its built-in admin panel simplified database management and debugging. Django's automatic URL routing system handled navigation within the application. Its support for RESTful APIs facilitated smooth communication with the frontend and machine learning models.

**Scikit-learn**

Scikit-learn provided machine learning algorithms for defect prediction. It supported the implementation of supervised learning models like Random Forest (RF), Support Vector Machine (SVM), and Naïve Bayes (NB). Scikit-learn's preprocessing tools, such as normalization and scaling, prepared data for model training. Its hyperparameter tuning functions allowed performance optimization. The library's cross-validation methods ensured unbiased performance evaluation. Built-in metrics like accuracy, F1-score, and precision facilitated model evaluation. Scikit-learn's ensemble methods like bagging and boosting allowed easy combination of multiple classifiers. Its modular nature supported easy model extension and customization. Scikit-learn's integration with other Python libraries enhanced data handling and visualization. Its active community provided regular updates and support.

**TensorFlow/Keras**

TensorFlow and Keras were used to build and train the Artificial Neural Network (ANN). TensorFlow's graph-based computational model optimized memory and processing efficiency. Keras simplified the creation of multi-layer neural networks using its high-level API. TensorFlow's GPU acceleration significantly improved training speed. Backpropagation and gradient descent algorithms helped adjust model weights for higher accuracy. Keras provided flexibility in defining activation functions and loss functions. TensorFlow's TensorBoard allowed real-time monitoring of model performance. Its compatibility with cloud platforms enabled distributed training. TensorFlow's regularization techniques prevented overfitting. Keras' ability to load pre-trained models simplified model refinement. Its support for the sequence of the models are allowed to handling the time-series of data.

**Pandas**

Pandas handled data manipulation and analysis tasks. Its Data Frame structure simplified data organization and processing. Pandas supported handling of missing values and outliers using its built-in functions. It enabled merging, grouping, and sorting of large datasets. Its compatibility with other libraries like NumPy and Matplotlib made data visualization easy. Pandas' efficient handling of time-series data improved performance in sequential data processing. It supported various file formats like CSV, Excel, and JSON for data import and export. Pandas' vectorized operations minimized processing time. Its indexing and slicing capabilities facilitated easy data retrieval. Pandas' integration with machine learning models enabled quick feature engineering and selection.

**NumPy**

NumPy provided foundational support for numerical computations. Its ndarray object allowed fast matrix operations and vectorized computations. NumPy's broadcasting feature simplified handling of different-shaped arrays. It supported linear algebra operations like matrix multiplication and eigenvalue computation. NumPy's random number generation assisted in data augmentation and shuffling. Its fast element-wise operations enhanced model training efficiency. NumPy's compatibility with TensorFlow and Scikit-learn ensured smooth data transfer between libraries. Its handling of multi-dimensional arrays allowed complex data modeling. NumPy's memory-mapped files enabled large dataset processing without overloading RAM. Its ability to integrate with low-level languages like C and Fortran ensured high-speed execution.

**Matplotlib & Seaborn**

Matplotlib and Seaborn were used for visualizing model performance and data insights. Matplotlib's Pyplot API allowed quick creation of line, bar, and scatter plots. Seaborn provided additional styling options and built-in statistical plotting functions. Visualization of accuracy, precision, recall, and F1-score simplified model evaluation. Heatmaps and correlation matrices helped identify data relationships. Custom styling and labeling improved the readability of plots. Matplotlib's integration with Pandas enabled direct plotting from DataFrames. Seaborn's violin and box plots provided insights into data distribution. Interactive plots allowed real-time monitoring of model performance. Color mapping in Seaborn helped highlight outliers and anomalies. Both libraries supported saving figures in multiple formats for reporting.

**HTML (Hyper Text Markup Language)**

HTML is the foundational language used to create the structure and content of web pages. It defines various elements like headings, paragraphs, images, links, and tables, organizing them into a structured format. HTML documents are composed of a tree-like structure, where elements are nested within other elements. It allows the inclusion of multimedia content, such as videos and audio, and forms for user input. Modern HTML also supports semantic elements that improve search engine optimization (SEO) and accessibility. HTML5 introduced new features like video playback, geolocation, and local storage to enhance the functionality of web applications.

**CSS (Cascading Style Sheets)**

CSS is a stylesheet language used to control the visual presentation of HTML elements. It defines the layout, color, fonts, spacing, and responsiveness of a webpage. CSS allows developers to separate content from design, making it easier to maintain and update the visual style of a website. It supports advanced styling techniques such as animations, transitions, and keyframes to create interactive and engaging designs.

CSS frameworks like Bootstrap and Tailwind simplify the development process by providing pre-defined styles and responsive components. With media queries, CSS enables the creation of responsive designs that adapt to different screen sizes and devices.

### **JavaScript**

JavaScript is a versatile programming language used to add interactivity and dynamic behavior to web pages. It enables client-side scripting, allowing developers to create features such as form validation, real-time updates, and dynamic content rendering. JavaScript interacts with the DOM (Document Object Model) to modify HTML and CSS based on user actions. It supports asynchronous programming using promises and `async/await`, making it possible to handle data fetching and background tasks without blocking the user interface. JavaScript frameworks and libraries like React, Angular, and Vue.js simplify the development of complex web applications.

### **SQL (Structured Query Language)**

SQL is a domain-specific language used to manage and manipulate relational databases. It allows developers to create, retrieve, update, and delete data stored in databases. SQL follows a structured format based on tables, with rows representing records and columns representing attributes. It supports complex queries using `JOIN`, `GROUP BY`, and `ORDER BY` clauses to combine and organize data from multiple tables. SQL ensures data integrity through constraints like primary keys and foreign keys. Advanced features such as stored procedures, triggers, and indexing improve database performance and automate data management.