Web Engineering Notes for Finals

---

1. Introduction

Introduction of Web Engineering

Web Engineering is the process of designing, developing, testing, and maintaining web-based systems or applications. It focuses on principles, methodologies, and tools to ensure the reliability, scalability, and performance of web applications.

- **Example**: A team developing an online shopping website like Amazon would follow web engineering principles to handle large traffic and ensure smooth operations.

A Brief Introduction to the Internet

The Internet is a global network of interconnected devices that communicate using TCP/IP protocols. It enables information sharing and communication worldwide.

- **Example**: The act of sending an email, accessing social media, or streaming a video all rely on the Internet.

The World Wide Web (WWW)

The WWW is a system of interlinked hypertext documents accessed via the Internet. It was created by **Tim Berners-Lee** in 1991.

- **Example**: Websites like Wikipedia, Facebook, or YouTube are all part of the WWW.

Web vs Internet

- **Internet**: A network infrastructure connecting computers globally.
- **Web**: A service that uses the Internet to access information.
- **Analogy**: The Internet is like a road system, while the Web is like the vehicles traveling on those roads.

Web Browsers

Web browsers are software applications used to access and display web pages. Examples include Chrome, Firefox, Safari, and Edge.

- **Example**: When you type `www.google.com` in Chrome, it retrieves the web page from a server and displays it.

Web Servers

Web servers are software or hardware that store, process, and deliver web pages to browsers. Examples include Apache, NGINX, and IIS.

- **Example**: A web server hosting `www.amazon.com` sends the requested web page to your browser when you search for a product.

Uniform Resource Locators (URL)

A URL is an address used to access resources on the web.

- **Format**: `https://www.example.com/path/resource`
- **Components**:
  - **Protocol**: `https://`
  - **Domain name**: `www.example.com`
  - **Path**: `/path/resource`

Hypertext

Hypertext is text containing links (hyperlinks) that connect to other documents or resources.

- **Example**: A Wikipedia article with links to related topics.

The Hyper Text Transfer Protocol (HTTP)

HTTP is a protocol for transferring web pages over the Internet. It uses request-response cycles between clients (browsers) and servers.

- **Example**: Clicking a link sends an HTTP request to a server, which responds with the requested page.

IP Address

An IP address is a unique identifier for devices on a network.

- **Example**: `192.168.1.1` is an IPv4 address.

IPv4 vs IPv6

- **IPv4**: 32-bit addresses, limited to 4.3 billion devices. (e.g., `192.168.0.1`)
- **IPv6**: 128-bit addresses, virtually unlimited. (e.g., `2001:0db8:85a3:0000:0000:8a2e:0370:7334`)

Website vs Web Application

- **Website**: Static content meant for information. (e.g., Blogs)
- **Web Application**: Dynamic and interactive, performs functions. (e.g., Gmail, Facebook)

---

2. Domain Name Structure

A domain name is divided into several parts:

- **Example**: `mail.google.com`
  - **Top-Level Domain (TLD)**: `.com`
  - **Second-Level Domain**: `google`
  - **Subdomain**: `mail`

---

3. Domain Name Working

- **Step 1**: User types a domain name (e.g., `www.google.com`).
- **Step 2**: DNS (Domain Name System) translates the domain name into an IP address.
- **Step 3**: The browser sends a request to the server at the IP address.
- **Step 4**: Server responds with the web page.

---

4. Web Request – Response Cycle

1. **Client** sends an HTTP request to a server.
2. **Server** processes the request.
3. **Server** sends back an HTTP response.
4. **Client** (browser) displays the response.

---

5. Categories of Web Applications

Document-Centric Web Application

Static applications focused on displaying documents.

- **Example**: Online PDF viewers.

Interactive Web Application

Allows user interaction, like clicking buttons or submitting forms.

- **Example**: Google Forms.

Transactional Web Application

Handles transactions like purchasing items.

- **Example**: Amazon checkout process.

Workflow-Based Web Application

Applications that automate tasks.

- **Example**: Employee leave management systems.

Collaborative Web Application

Focuses on teamwork and collaboration.

- **Example**: Google Docs, Trello.

Portal-Oriented Web Application

Aggregates content from multiple sources.

- **Example**: Yahoo! News portal.

Ubiquitous Web Application

Accessible from anywhere.

- **Example**: Mobile-friendly Gmail.

Knowledge-Based Web Application

Focused on information sharing.

- **Example**: Wikipedia.

---

6. Web Application Architecture

Single Tier

All logic and data are on a single machine.

- **Example**: Simple static websites.

Client-Server (Two Tier)

Data is stored on a server, and clients request data.

- **Example**: Email services.

Three Tier

1. **Presentation Tier**: User interface (HTML/CSS).
2. **Logic Tier**: Business logic (Backend).
3. **Data Tier**: Database storage.

---

7. Introduction to HTML

HTML (HyperText Markup Language) structures web pages.

Creating an HTML Document

Basic structure:

```
<!DOCTYPE html>
<html>
<head>
   <title>My Page</title>
</head>
<body>
   <h1>Hello World</h1>
</body>
</html>
```

Nesting HTML Elements

Elements can be nested within each other.

- **Example**:

```
<p>This is a <strong>nested</strong> element.</p>
```

Head Elements & Scripts in HTML

`<head>` includes metadata and scripts.

```
<head>
   <script src="script.js"></script>
</head>
```

HTML Layout Elements

Elements for layout: `<header>`, `<main>`, `<footer>`.

Embedding Audios & Videos

```
<video controls>
   <source src="video.mp4" type="video/mp4">
</video>
```

Navbar & List Items

```
<ul>
   <li>Home</li>
   <li>About</li>
</ul>
```

Headings in HTML

`<h1>` to `<h6>` for headings.

HTML Paragraphs & Text Formatting

`<p>`, `<strong>`, `<em>`.

HTML Table

```
<table>
   <tr><th>Name</th><th>Age</th></tr>
   <tr><td>John</td><td>25</td></tr>
</table>
```

Page Linking

```
<a href="page2.html">Go to Page 2</a>
```

Forms & Inputs

```
<form>
   <input type="text" placeholder="Name">
   <input type="submit">
</form>
```

Inline & Block Elements

- **Inline**: `<span>`, `<a>`.
- **Block**: `<div>`, `<p>`.

CSS

Here's the **detailed breakdown** for **CSS and Tailwind** with explanations, examples, and scenarios to ensure you understand the topics thoroughly for exams and viva.

---

1. Introduction to CSS

CSS (**Cascading Style Sheets**) is used to control the layout, design, and appearance of HTML elements on a webpage.

CSS Documents & The Cascade

- **CSS Documents**: Styles can be added using:
  1. **Inline CSS**: Directly in an HTML element using `style`.
  2. `<p style="color: red;">This is red text</p>`
  3. **Internal CSS**: Inside `<style>` tags in the HTML `<head>`.
  4. `<style>`
  5. `  p { color: blue; }`
  6. `</style>`
  7. **External CSS**: In a separate file (e.g., `style.css`) linked via `<link>`.
  8. `<link rel="stylesheet" href="style.css">`
- **The Cascade**: CSS prioritizes styles based on specificity, source, and importance. **Order of Precedence**: Inline → Internal → External.

Selectors, Properties & Values

- **Selectors** target HTML elements.
  - **Example**:
  - `p { color: green; font-size: 16px; }`
    - **p** → Selector
    - **color** and **font-size** → Properties
    - **green** and **16px** → Values

Classes & IDs

- **Classes**: Reusable for multiple elements, prefixed with `.`.
- `.button { background: blue; color: white; }`
- `<div class="button">Click Me</div>`

- **IDs**: Unique, prefixed with `#`.
- `#header { font-size: 20px; }`

Specificity in CSS

CSS calculates priority when styles conflict:

1. Inline styles have the highest specificity.
2. IDs are more specific than classes.
3. Element selectors have the lowest priority.

- **Example**:
- `<p id="text" class="red-text">Specificity Example</p>`
- `#text { color: blue; }       /* Wins because of ID */`
- `.red-text { color: red; }    /* Loses */`

Setting Widths & Heights

Defines the size of elements.

- **Example**:
- `div { width: 200px; height: 100px; }`

Length Units

- **Absolute Units**: `px`, `cm`, `mm`.
- **Relative Units**: `em`, `rem`, `%`, `vw`, `vh`.
- `p { font-size: 2em; }  /* 2 times the parent font size */`

Colors & Color Types

- Types of colors:
    1. **Named colors**: `red`, `blue`.
    2. **Hex**: `#ff0000` (Red).
    3. **RGB**: `rgb(255, 0, 0)`.
    4. **RGBA**: `rgba(255, 0, 0, 0.5)` (50% transparent).

Padding, Margin & Borders

- **Padding**: Space between content and border.

- **Margin**: Space outside the border.
- **Borders**: Visual boundaries around content.
- `div {`
- `  margin: 20px;`
- `  padding: 10px;`
- `  border: 2px solid black;`
- `}`

The Box Model

All elements follow the box model:

1. **Content**
2. **Padding**
3. **Border**
4. **Margin**

Visibility

Controls whether an element is visible.

- **Example**:
- `.hidden { visibility: hidden; } /* Takes up space but invisible */`

Working with Fonts

Sets fonts for text:

```
p { font-family: Arial, sans-serif; font-size: 16px; }
```

Element Flow (Block & Inline)

- **Block Elements**: Start on a new line (e.g., `<div>`, `<p>`).
- **Inline Elements**: Do not break lines (e.g., `<span>`, `<a>`).

Float Layout

Moves elements to the left/right for layouts.

```
img { float: left; margin-right: 10px; }
```

Position Property

Controls positioning:

- **Static**: Default.
- **Relative**: Positions relative to its normal flow.
- **Absolute**: Positions relative to the nearest ancestor.
- **Fixed**: Fixed on screen.

CSS Pseudo Classes

Add styles when an element is in a specific state.

- **Example**:
- `a:hover { color: red; } /* Change color on hover */`

Grid

CSS Grid is for creating two-dimensional layouts.

- **Example**:
- `.grid-container {`
- `  display: grid;`
- `  grid-template-columns: 1fr 2fr;`
- `}`

Flexbox

Flexbox aligns elements in one dimension (row or column).

- **Example**:
- `.flex-container {`
- `  display: flex;`
- `  justify-content: center; /* Align items horizontally */`
- `  align-items: center; /* Align items vertically */`
- `}`

---

2. Introduction to Tailwind CSS

Tailwind is a utility-first CSS framework for rapidly building designs.

Installation

Install via npm:

```
npm install tailwindcss
```

Core Concepts

Utility Classes

Tailwind uses small utility classes instead of writing custom CSS.

- **Example**:
- `<div class="text-blue-500 bg-gray-200 p-4">Hello Tailwind</div>`

Basic Structure

Typography

Control font size and text color.

- **Example**:
- `<p class="text-lg text-gray-700">Large Text</p>`

Spacing (Margin & Padding)

- **Example**:
- `<div class="m-4 p-2">Margin and Padding</div>`

Colors

Tailwind provides color utilities:

- **Example**:
- `<div class="bg-red-500 text-white">Red Background</div>`

Commonly Used Utilities

Text Alignment

- **Example**:
- `<p class="text-center">Centered Text</p>`

Fonts

Tailwind provides utilities for font weights and styles.

```
<p class="font-bold italic">Bold Italic</p>
```

Background Colors

```
<div class="bg-blue-200">Light Blue Background</div>
```

Borders & Rounded Corners

```
<div class="border border-gray-400 rounded-lg">Border Example</div>
```

Flexbox & Layout

```
<div class="flex justify-between">
   <div>Item 1</div>
   <div>Item 2</div>
</div>
```

Width & Height

```
<div class="w-1/2 h-32 bg-gray-300">Half Width, Fixed Height</div>
```

Display

Control element visibility:

```
<div class="hidden">This is hidden</div>
```

Responsive Design

Breakpoints

Tailwind uses `sm`, `md`, `lg`, and `xl` for breakpoints.

- **Example**:
- `<div class="text-base md:text-xl lg:text-2xl">Responsive Text</div>`

Hover & Focus

Tailwind provides utilities for interactive states.

- **Example**:
- `<button class="bg-blue-500 hover:bg-blue-700 focus:outline-none">Hover Me</button>`

This detailed guide covers both **CSS** and **Tailwind CSS** with examples for each topic to ensure clear understanding. Let me know if you need additional clarifications! 🚀

JavaScript

Here's the detailed breakdown of **JavaScript topics** for your exam and viva preparation, including explanations, examples, and scenarios for better understanding:

---

1. Introduction to JavaScript

JavaScript is a **client-side scripting language** used to create dynamic, interactive websites. It can also be used for server-side programming with Node.js.

Variables & Const Variables

- `var`, `let`, and `const` are used to declare variables.
  - `let` → Block-scoped.
  - `const` → Cannot be reassigned.
- **Example**:
- `let name = "John";`
- `const age = 30;`

Numbers & Strings

JavaScript handles **numbers** and **strings**.

- **Example**:
- `let num = 10;  // Number`
- `let text = "Hello World";  // String`

Booleans

True/False values used for logical decisions.

- **Example**:
- `let isAdmin = true;`

Type Conversion

parseInt & parseFloat

Convert strings to numbers.

- **Example**:
- `let num = parseInt("123");   // 123`
- `let floatNum = parseFloat("12.34"); // 12.34`

Number() or + (Short Hand)

- **Example**:
- `let value = "45";`
- `let num = Number(value);  // 45`
- `let shorthand = +value;   // 45`

Arrays

Arrays store multiple values.

2D, 3D, Multi-dimensional Arrays

- **Example**:
- `let arr = [[1, 2], [3, 4]];  // 2D Array`
- `let multi = [[[1], [2]], [[3], [4]]]; // 3D Array`

Heterogeneous Arrays

Arrays can hold different types of values.

- **Example**:
- `let mixed = [1, "text", true];`

Objects (Simple and Nested)

- **Simple Objects**:
- `let person = { name: "John", age: 30 };`
- **Nested Objects**:
- `let user = {`
- `  name: "Alice",`
- `  address: { city: "New York", zip: 12345 }`
- `};`

Operators

- **Arithmetic**: `+`, `-`, `*`, `/`, `%`

- **Relational**: <, >, <=, >=
- **Increment/Decrement**: ++, --
- **Example**:
- `let x = 5;`
- `x++;`
- `console.log(x); // 6`

If, Else-if, And & Or

- **Example**:
- `let age = 18;`
- `if (age > 18) {`
- `  console.log("Adult");`
- `} else if (age === 18) {`
- `  console.log("Just turned adult");`
- `} else {`
- `  console.log("Minor");`
- `}`

Switch Statement

Used for multiple conditions.

- **Example**:
- `let day = 2;`
- `switch(day) {`
- `  case 1: console.log("Monday"); break;`
- `  case 2: console.log("Tuesday"); break;`
- `  default: console.log("Other Day");`
- `}`

Loops

For Loop

```
for (let i = 0; i < 5; i++) {
   console.log(i);
}
```

While Loop

```
let i = 0;
while (i < 5) {
   console.log(i);
   i++;
}
```

Do While Loop

```
let i = 0;
```

```
do {
   console.log(i);
   i++;
} while (i < 5);
```

For…of (Iterates over values)

```
let arr = [1, 2, 3];
for (let num of arr) {
   console.log(num);
}
```

For…in (Iterates over keys)

```
let obj = { a: 1, b: 2 };
for (let key in obj) {
   console.log(key, obj[key]);
}
```

Scope

- **Function-level**: `var`
- **Block-level**: `let` and `const`
- **Global**: Accessible everywhere.

Functions

Arrow Functions

```
const add = (a, b) => a + b;
```

Anonymous Functions

```
const greet = function() {
   console.log("Hello");
};
```

Error & its Types

1. **Syntax Error**: Invalid code syntax.
2. **Type Error**: Invalid operation on a type.
3. **Range Error**: Exceeding valid range.
4. **Reference Error**: Using undefined variables.

Exception Handling

Throwing Strings, Numbers, Objects, and `new Error`

```
try {
   throw new Error("Custom Error");
} catch (e) {
```

```
   console.error(e.message);
}
```

High Order Functions

Functions that take other functions as arguments.

- **Example**:
- `const arr = [1, 2, 3];`
- `arr.forEach(num => console.log(num));`

Closures

A function that remembers the environment where it was created.

- **Example**:
- `function outer() {`
- `   let count = 0;`
- `   return function inner() {`
- `      count++;`
- `      console.log(count);`
- `   };`
- `}`
- `const counter = outer();`
- `counter(); // 1`
- `counter(); // 2`

Iterators

- `.forEach()`: Executes a function for each element.
- `.map()`: Returns a new array.
- `.filter()`: Filters elements.
- `.find()`: Finds the first match.
- `.findIndex()`: Returns the index of the first match.
- `.reduce()`: Reduces to a single value.
- `.some()`: Checks if any element matches.
- `.every()`: Checks if all elements match.
- `.sort()`: Sorts elements.

Example of `.map` and `.filter`:

```
let nums = [1, 2, 3, 4];
let squared = nums.map(n => n * n);
let even = nums.filter(n => n % 2 === 0);
```

---

2. Introduction to DOM

DOM (Document Object Model) allows interaction with HTML elements.

Access Elements in the DOM

- `querySelector`, `getElementById`, `getElementsByTagName`.
- `let elem = document.querySelector("#myElement");`

---

3. Working with Events

- **Adding/Removing Event Listeners**:
- `button.addEventListener("click", () => alert("Clicked"));`
- **`preventDefault()`**: Stops default behavior (e.g., form submission).

---

4. Web Working

Fetch API (GET & POST):

```
fetch("https://api.example.com/data")
   .then(res => res.json())
   .then(data => console.log(data));
```

---

5. ES6 Features

- **Template Literals**:
- `let text = `Hello, ${name}`;`
- **Destructuring**:
- `let { name, age } = { name: "John", age: 30 };`
- **Promises**:
- `const myPromise = new Promise((resolve, reject) => {`
- `   resolve("Success!");`
- `});`
- **Async/Await**:
- `async function fetchData() {`
- `   let response = await fetch("url");`
- `   let data = await response.json();`
- `   console.log(data);`
- `}`

This **structured breakdown** with examples and explanations will help you master JavaScript for exams and viva. Let me know if you need further explanations or examples! 🚀

React

Here's a structured and detailed breakdown of **React.js topics** to help you prepare effectively for your exams and projects:

---

1. Introduction to React.js

React.js is a **JavaScript library** for building fast and interactive UIs. It is component-based and uses a virtual DOM for efficient updates.

Core Features

- **Component-based architecture**: Reusable UI components.
- **Declarative**: Code describes *what to render*, not *how*.
- **Virtual DOM**: React creates a virtual representation of the UI for faster updates.
- **One-way data binding**: Ensures predictable data flow.

Thinking in React.js

Steps to build a React app:

1. Break the UI into **components**.
2. Build a static version using **props**.
3. Identify **state** and make components dynamic.
4. Identify where **state** lives.
5. Add interactivity using **events**.

Building an App Using Components

Components are the building blocks of React apps.

- **Functional Components**: Use `function` and hooks.
- **Class Components**: Use `class` with `render()`.
- **Example** (Functional Component):
- ```
  function App() {
  ```
- ```
      return <h1>Hello World!</h1>;
  ```

- `}`

ReactDOM, JSX & Babel

- **ReactDOM**: Renders React components to the DOM.
- `ReactDOM.render(<App />, document.getElementById("root"));`
- **JSX**: JavaScript XML syntax for writing HTML in React.
- `const element = <h1>Hello, JSX!</h1>;`
- **Babel**: Converts JSX into regular JavaScript.

Creating Custom Components

Custom components are created using functions or classes:

```
function Greeting(props) {
   return <h2>Hello, {props.name}!</h2>;
}
```

Introduction to Props

Props are arguments passed to components. They are **read-only**.

```
function Welcome(props) {
   return <h1>Welcome, {props.user}</h1>;
}
ReactDOM.render(<Welcome user="John" />, document.getElementById("root"));
```

Adding CSS Classes to JSX

Use `className` instead of `class`:

```
function Box() {
   return <div className="box">Styled Box</div>;
}
```

Conditional JSX

Conditionally render components using `if`, ternary, or logical operators:

```
function Greeting({ isLoggedIn }) {
   return isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Login</h1>;
}
```

Using Babel in Production

For production, **Babel** compiles JSX and modern JavaScript into a browser-compatible version.

---

2. Webpack

Introduction to Webpack

Webpack bundles JavaScript, CSS, and assets into a single file.

Create Hello World App

Steps:

1. Install Webpack and dependencies (`npm install webpack webpack-cli --save-dev`).
2. Create `index.js` and `index.html`.
3. Configure Webpack in `webpack.config.js`.

- **Example Webpack Config**:
- ```
  module.exports = {
     entry: './src/index.js',
     output: {
        filename: 'bundle.js',
        path: __dirname + '/dist',
     },
     mode: 'development',
  };
  ```

---

3. Props

Usage of Props

Props allow data to flow from **parent to child** components.

PropTypes and DefaultProps

- **PropTypes**: Validate props.
- ```
  import PropTypes from 'prop-types';
  ```

```
function Greeting({ name }) {
   return <h1>Hello, {name}</h1>;
}

Greeting.propTypes = {
   name: PropTypes.string.isRequired,
};
```

- **DefaultProps**: Provide default values for props.

```
Greeting.defaultProps = {
   name: "Guest",
};
```

---

4. State

Important State Concepts

State allows components to manage data that changes over time.

Pure Functions & setState

Always use **setState** to update state.

- **Example**:

```
import { useState } from 'react';

function Counter() {
   const [count, setCount] = useState(0);
   return (
      <button onClick={() => setCount(count + 1)}>
         Clicked {count} times
      </button>
   );
}
```

Passing State to Child and Parent Components

- **Child**: Pass state as a prop.
- **Parent**: Use callbacks to pass state back.

---

5. Component Architecture

Component Hierarchy

Organize components into a parent-child tree.

Declarative vs Imperative

- **Declarative**: Tell React *what to do* (React way).
- **Imperative**: Directly manipulate DOM.

Virtual DOM

React updates the virtual DOM first, then synchronizes with the real DOM for efficiency.

---

6. Events in React

Use `onClick`, `onChange`, etc.

```
function Button() {
   return <button onClick={() => alert("Clicked!")}>Click Me</button>;
}
```

---

7. Fetching Data from an API

Use `fetch()` or Axios.

- **Example**:
- `import { useEffect, useState } from 'react';`
-
- `function Data() {`
- `   const [data, setData] = useState([]);`
-
- `   useEffect(() => {`
- `      fetch("https://api.example.com/posts")`
- `         .then(res => res.json())`
- `         .then(data => setData(data));`
- `   }, []);`
-
- `   return <ul>{data.map(item => <li key={item.id}>{item.title}</li>)}</ul>;`
- `}`

---

8. Forms and Refs

- **Forms**: Controlled components using `useState`.

- `const [value, setValue] = useState("");`
-
- `<input value={value} onChange={(e) => setValue(e.target.value)} />;`
- **Refs**: Access DOM elements directly.

---

9. Component Lifecycle Methods (Class Components)

- **Mounting**: `componentDidMount()`
- **Updating**: `componentDidUpdate()`
- **Unmounting**: `componentWillUnmount()`

---

10. Hooks Introduction

Hooks add state and lifecycle methods to functional components.

- Common Hooks: `useState`, `useEffect`, `useContext`, `useRef`.

---

11. To-Do App

Build a simple app to manage tasks using state, props, and events.

---

12. React Router

For single-page applications (SPA).

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';

<BrowserRouter>
   <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
   </Routes>
</BrowserRouter>
```

---

13. Debugging React App

JEST

For unit testing React components.

Enzyme

Enables testing of React components' output and behavior.

This structured explanation covers everything, including examples and concepts for **React.js, Webpack, State Management, and Lifecycle Methods**. Let me know if you need examples for specific concepts or additional practice projects! 🚀

The End