Expt 8: Change specification and use any SCM Tool to make different versions

AIM: Students will able to use SCM Tool to make handle versioning of projects

Theory

Software configuration management: The traditional software configuration management (SCM) process is looked upon by practitioners as the best solution to handling changes in software projects. It identifies the functional and physical attributes of software at various points in time, and performs systematic control of changes to the identified attributes for the purpose of maintaining software integrity and traceability throughout the software development life cycle.

The SCM process further defines the need to trace changes, and the ability to verify that the final delivered software has all of the planned enhancements that are supposed to be included in the release. It identifies four procedures that must be defined for each software project to ensure that a sound SCM process is implemented. They are:

1. Configuration identification
2. Configuration control
3. Configuration status accounting
4. Configuration audits

These terms and definitions change from standard to standard, but are essentially the same.

- Configuration identification is the process of identifying the attributes that define every aspect of a configuration item. A configuration item is a product (hardware and/or software) that has an end-user purpose. These attributes are recorded in configuration documentation and baselined. Baselining an attribute forces formal configuration change control processes to be effected in the event that these attributes are changed.
- Configuration change control is a set of processes and approval stages required to change a configuration item's attributes and to re-baseline them.
- Configuration status accounting is the ability to record and report on the configuration baselines associated with each configuration item at any moment of time.
- Configuration audits are broken into functional and physical configuration audits. They occur either at delivery or at the moment of effecting the change. A functional configuration audit ensures that functional and performance attributes of a configuration item are achieved, while a physical configuration audit ensures that a configuration item is installed in accordance with the requirements of its detailed design documentation.

GitHub offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command- line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management for every project.

Result and Discussion:

Version 1 of BMS:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Banking App</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f0f2f5;
      display: flex;
      flex-direction: column;
      align-items: center;
      margin-top: 50px;
    }
    .container {
      background: white;
      padding: 20px 40px;
      border-radius: 10px;
      box-shadow: 0 0 15px rgba(0,0,0,0.1);
      width: 300px;
    }
    h2 {
      text-align: center;
    }
    .balance {
      font-size: 1.5em;
      margin: 20px 0;
      text-align: center;
    }
    input[type="number"] {
      width: 100%;
      padding: 8px;
      margin: 10px 0;
      box-sizing: border-box;
    }
    button {
      width: 100%;
      padding: 10px;
      margin-bottom: 10px;
      border: none;
      border-radius: 5px;
      background-color: #4CAF50;
      color: white;
      font-size: 1em;
```

```
<body>
  <div class="container">
    <h2>Banking App</h2>
    <div class="balance">Balance: $<span
id="balance">0.00</span></div>

    <input type="number" id="amount"
placeholder="Enter amount">
    <button
onclick="deposit()">Deposit</button>
    <button
onclick="withdraw()">Withdraw</button>

    <div class="history">
      <h3>Transaction History</h3>
      <ul id="history"></ul>
    </div>
  </div>

  <script>
    let balance = 0;
    const balanceDisplay =
document.getElementById('balance');
    const historyList =
document.getElementById('history');

    function updateBalance() {
      balanceDisplay.textContent =
balance.toFixed(2);
    }

    function addToHistory(type, amount) {
      const li = document.createElement('li');
      li.textContent = `${type}:
$${amount.toFixed(2)}`;
      historyList.prepend(li); // most recent on top
    }

    function deposit() {
      const amountInput =
document.getElementById('amount');
      const amount =
parseFloat(amountInput.value);
      if (isNaN(amount) || amount <= 0) {
```

```
      cursor: pointer;
    }
    button:hover {
      background-color: #45a049;
    }
    .history {
      margin-top: 20px;
    }
    .history h3 {
      margin-bottom: 10px;
    }
    ul {
      list-style-type: none;
      padding: 0;
      font-size: 0.9em;
    }
  </style>
</head>
```

```
        alert('Please enter a valid amount.');
        return;
      }
      balance += amount;
      updateBalance();
      addToHistory('Deposited', amount);
      amountInput.value = '';
    }

    function withdraw() {
      const amountInput =
document.getElementById('amount');
      const amount =
parseFloat(amountInput.value);
      if (isNaN(amount) || amount <= 0) {
        alert('Please enter a valid amount.');
        return;
      }
      if (amount > balance) {
        alert('Insufficient funds!');
        return;
      }
      balance -= amount;
      updateBalance();
      addToHistory('Withdrew', amount);
      amountInput.value = '';
    }
  </script>
</body>
</html>
```

Version 2 of BMS:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Enhanced Banking App v2</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: #e3f2fd;
      display: flex;
      justify-content: center;
      margin-top: 60px;
    }
    .container {
```

```
   <h2>Banking App v2 (Enhanced)</h2>
    <div class="balance">Balance: $<span
id="balance">0.00</span></div>

    <input type="number" id="amount"
placeholder="Enter amount">
    <button onclick="deposit()">Deposit</button>
    <button
onclick="withdraw()">Withdraw</button>

    <div class="history">
      <h3>Transaction History</h3>
      <ul id="history"></ul>
    </div>
```

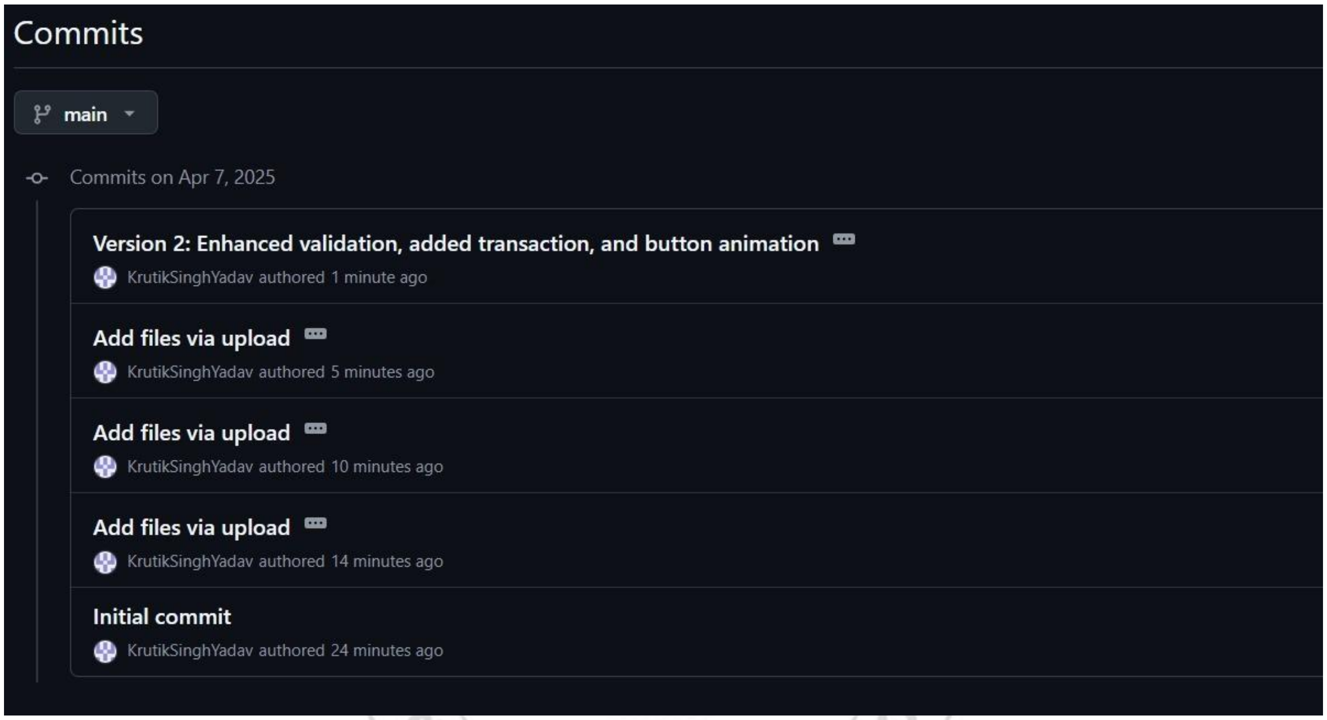
```
      background: #fff;
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 10px 25px rgba(0, 0, 0, 0.1);
      width: 400px;
    }
    h2 {
      text-align: center;
      color: #1976d2;
    }
    .balance {
      font-size: 2em;
      margin: 20px 0;
      text-align: center;
    }
    input[type="number"] {
      width: 100%;
      padding: 10px;
      margin: 15px 0;
      border: 1px solid #ccc;
      border-radius: 6px;
    }
    button {
      width: 100%;
      padding: 10px;
      margin-bottom: 10px;
      border: none;
      border-radius: 6px;
      background-color: #1976d2;
      color: white;
      font-size: 1em;
      cursor: pointer;
      transition: transform 0.1s ease-in-out, background-color 0.3s;
    }
    button:hover {
      background-color: #1565c0;
    }
    button:active {
      transform: scale(0.98);
    }
    .history {
      margin-top: 20px;
    }
    .history h3 {
```

```
  </div>

  <script>
    let balance = parseFloat(localStorage.getItem("balance")) || 0;
    let transactions = JSON.parse(localStorage.getItem("transactions")) || [];

    const balanceDisplay = document.getElementById('balance');
    const historyList = document.getElementById('history');
    const amountInput = document.getElementById('amount');

    function updateUI() {
      balanceDisplay.textContent = balance.toFixed(2);
      historyList.innerHTML = "";
      transactions.slice().reverse().forEach((t, index) => {
        const li = document.createElement('li');
        li.innerHTML = `${t.type}: $${t.amount.toFixed(2)}<br><small>${t.date}</small>
        <button class="delete-btn" onclick="deleteTransaction(${transactions.length - 1 - index})">🗑</button>`;
        historyList.appendChild(li);
      });
    }

    function saveData() {
      localStorage.setItem("balance", balance);
      localStorage.setItem("transactions", JSON.stringify(transactions));
    }

    function validateAmount(value) {
      if (value === '' || isNaN(value) || value <= 0) {
        alert('Please enter a valid amount greater than 0.');
        return false;
      }
      return true;
    margin-bottom: 10px;
  }
  ul {
    list-style-type: none;
    padding: 0;
    max-height: 200px;
    overflow-y: auto;
  }
  li {
    padding: 8px;
    margin-bottom: 6px;
    border-radius: 6px;
    background: #f9f9f9;
    border-left: 5px solid #1976d2;
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-size: 0.95em;
  }
  .delete-btn {
    background: none;
    border: none;
    color: #d32f2f;
    font-size: 1em;
    cursor: pointer;
    margin-left: 10px;
  }
  .delete-btn:hover {
    color: #b71c1c;
  }
 </style>
</head>
<body>
 <div class="container">
```

```
  }

  function deposit() {
    const amount = parseFloat(amountInput.value);
    if (!validateAmount(amount)) return;

    balance += amount;
    transactions.push({ type: 'Deposited', amount,
date: new Date().toLocaleString() });
    saveData();
    updateUI();
    amountInput.value = '';
  }

  function withdraw() {
    const amount = parseFloat(amountInput.value);
    if (!validateAmount(amount)) return;

    if (amount > balance) {
      alert('Insufficient funds!');
      return;
    }

    balance -= amount;
    transactions.push({ type: 'Withdrew', amount,
date: new Date().toLocaleString() });
    saveData();
    updateUI();
    amountInput.value = '';
  }

  function deleteTransaction(index) {
    const transaction = transactions[index];

    if (transaction.type === 'Deposited') {
      balance -= transaction.amount;
    } else if (transaction.type === 'Withdrew') {
      balance += transaction.amount;
    }

    transactions.splice(index, 1);
    saveData();
    updateUI();
  }

  // Initialize on load
```

	updateUI();
	  </script>
	</body>
	</html>

SCM using Git and GitHub: Version history and commits



Commits

main

Commits on Apr 7, 2025

Version 2: Enhanced validation, added transaction, and button animation
KrutikSinghYadav authored 1 minute ago

Add files via upload
KrutikSinghYadav authored 5 minutes ago

Add files via upload
KrutikSinghYadav authored 10 minutes ago

Add files via upload
KrutikSinghYadav authored 14 minutes ago

Initial commit
KrutikSinghYadav authored 24 minutes ago

This Git commit history outlines the progressive development of a Simple Banking Management System built using HTML, CSS, and JavaScript. The project started with the initial setup and basic file uploads that formed Version 1, which included fundamental operations like deposits and withdrawals with a simple UI.
As the project evolved, Version 2 was introduced with significant improvements:

- Enhanced form validation to prevent invalid inputs
- Transaction history log with timestamp tracking
- Edit and delete functionality for transactions
- Animated buttons for a better user experience
- Local storage integration to retain data after page reloads

The structured commit messages and versioning demonstrate good development practices and a clear upgrade path, making the project easier to maintain, improve, and scale.

Conclusion:

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				