* **Methodology / Implementation**

The development of the Crowdfunding DApp follows a modular, phased approach combining smart contract development, frontend integration, and blockchain deployment.

**Step 1: Smart Contract Development**

* **Write Smart Contracts** in Solidity to handle:
  + Campaign creation.
  + Accepting and tracking contributions.
  + Automatic goal verification and fund release/refund.
  + Optional milestone-based fund distribution.
* **Tools Used**: Hardhat or Truffle for development, testing, and compilation.

**Step 2: Smart Contract Deployment**

* Deploy the smart contract on a blockchain network:
  + First on a **testnet** like Goerli/Sepolia for testing.
  + Later on the **Ethereum Mainnet** or a Layer-2 network for production.
* **Deployment Tools**: Infura, Alchemy, MetaMask.

**Step 3: Frontend Development**

* Build a user-friendly web interface using **React.js** or **Next.js**.
* Connect the frontend with the blockchain through **Web3.js** or **Ethers.js** libraries.
* Key frontend functionalities:
  + Launch new campaigns.
  + Contribute to active campaigns.
  + Track funding status in real-time.
  + Claim refunds if the campaign fails.

**Step 4: Wallet and Payment Integration**

* Integrate MetaMask (or WalletConnect) to allow users to:
  + Connect their crypto wallets.
  + Approve transactions (funding, refunds) securely.

**Step 5: Testing**

* Write **unit tests** for smart contracts using Mocha/Chai to ensure:
  + Correct fund handling.
  + Secure and tamper-proof operations.
* Perform **frontend testing** to ensure smooth user interaction.

**Step 6: Deployment and Hosting**

* Host the frontend DApp using platforms like **Vercel**, **Netlify**, or **IPFS**.
* Connect the hosted frontend to the deployed smart contract.

**Step 7: Monitoring and Maintenance**

* Use blockchain explorers (e.g., Etherscan) to monitor smart contract activity.
* Plan upgrades if needed using proxy patterns (for upgradable contracts).