A smart wallet is a next-generation digital wallet that combines traditional payment functionality with advanced blockchain technology and programmable features. Unlike basic cryptocurrency wallets that simply store and transfer digital assets, smart wallets can execute complex operations through smart contracts, automate transactions, and interact directly with decentralized applications. They offer enhanced security through features like multi-signature authentication, social recovery options, and customizable spending limits. Smart wallets also enable users to manage multiple cryptocurrencies, tokens, and NFTs in one interface, while providing advanced features like account abstraction, gas fee optimization, and batch transactions. This technology is revolutionizing how users interact with Web3 platforms by simplifying complex blockchain operations and offering a more user-friendly experience comparable to traditional financial services.

P1: Identify functional and non-functional requirements from the system description.

- Notational: 1-0/17 = 1
  - Functional, Non-Functional
  - User Interface non-functional
- Contextual: 1-1/17 = 0.941
  - 10 Functional: Smart Contract Execution, Automated Transactions, Decentralized Application Interaction, Multi-Signature Authentication, Social Recovery Options, Customizable Spending Limits, Multi-Asset Management, Account Abstraction, Gas Fee Optimization, Batch Transactions.
  - 7 Non-functional: Security, User-Friendliness, Simplification, Interoperability, Scalability, Reliability, Performance.
  - Reliability not explicitly mentioned.

P2: Create a use case model for the system.

- Notational: 1-1/5 = 0.8
  - System Boundary (0.5), Actors (1.5), Use Cases (2), Relationships (1)
  - dApps and Blockchain Network actors not being used. (-1)
- Contextual: 1-(1+1*0.2)/10 = 0.88
  - 10 use cases: 1. Set Up Smart Wallet, 2. Deposit Cryptocurrency, 3. Execute Smart Contract, 4. Automate Transaction, 5. Interact with dApps, 6. Manage Multiple Assets, 7. Optimize Gas Fees, 8. Recover Account, 9. Set Custom Spending Limits, 10. Perform Batch Transactions
  - Set Up Smart Wallet not explicitly covered.
  - Invalid actor in UC7

P3: Create use case specifications for identified use cases.

- Notational: 1-(1+1*0.3)/6.5 = 0.8
  - Use Case Name (0.5), Actor (1), Precondition (1), Postcondition (1), Main Scenario (2), Alternative Scenario (1)
  - Missing actor (-1)
  - Alternative scenario having no reference to basic flow.
- Contextual: 1-0/3 = 1
  - 3 UC Specs: Set Up Smart Wallet, Deposit Cryptocurrency, Execute Smart Contract

P4: Create a domain model based on use case specifications.

- Notational: 1-1/5 = 0.8
  - Classes (2), Attributes (1), Relationships (1), Multiplicities (1)
  - Must not have Data Types (-0.5), Operations (-0.5), Navigabilities (-0.5), Visibilities (-0.5)
  - No multiplicities (-1)
- Contextual: 1-0/9 = 1
  - 9 classes: User, Wallet, MultipleAssets, SmartContract, dApp, DepositAddress, Transaction, Amount, Cryptocurrency
  - 7 relationships

P5: Identify system operations from use case specifications.

- Notational: 1-(0.2+0.2+0.2)/1.6 = 0.625
  - Base Use Case (0.2), Operation Name (1), Parameters (0.2), Return (0.2)
  - No base use case, no parameters, no return.
- Contextual: 1-11/20 = 0.45
  - 20 system operations: createUser, importWallet, restoreWallet, setPassword, setBiometricAuth, createWallet, getWalletBalance, updateWalletSettings, depositAsset, getAssetBalance, transferAsset, executeSmartContract, createSmartContract, getSmartContract, getSmartContractStatus, createTransaction, getTransaction, getTransactionHistory, confirmTransaction, interactWithDapp, getDappList, getDappDetails
  - 11 not covered in UC specs: createUser, importWallet, restoreWallet , setBiometricAuth, createTransaction, getTransaction, getTransactionHistory, confirmTransaction, interactWithDapp, getDappList, getDappDetails

P6: Create design sequence diagrams for system operations.

- Notational: 1-(0.2+0.2)/4.6 = 0.913
  - Base System Operation (0.2), Participants (1), Operations (1), Parameters (0.2), Return Type (0.2), Sequence of Operation Calls (2)
  - No parameters (0.2), no return type (0.2)
- Contextual: 1-(3*0.6)/5 = 0.64
  - 5 DSDs: createUser, depositAsset, executeSmartContract, interactWithDapp, getTransactionHistory
  - Not DSDs, but SSDs.

P7: Create design class diagrams based on the domain model and sequence diagrams.

- Notational: 1-(0.5*0.2+0.2+0.2+1+0.5+0.5)/5.2 = 0.519
  - Classes(1), Attributes with Types(0.5), Operations (1), Parameters with Types(0.2), Return Type (0.2), Relationship(1), Multiplicities(0.5), Navigabilities(0.5), Visibilities(0.3)
  - No data types, no parameters, no return types, no relationships, no multiplicities, no navigabilities
- Contextual: 1-(2*0.5+4)/12 = 0.583
  - 12 classes: User, UserManager, Wallet, WalletManager, Asset, AssetManager, SmartContract, SmartContractManager, Transaction, TransactionManager, Dapp, DappManager
  - 0 Relationships
  - Manager classes having duplicate methods (2 partial and 4 full) with the managed classes, which are not necessary.

P8: Develop a Java implementation for the system as specified in the class diagram and sequence diagrams.

- Notational: 1-(0.2+3)/5 = 0.36
  - Classes (0.5), Data types (0.5), Visibilities (0.5), Constructor (0.1), Getters (0.2), Setters (0.2), Methods (3)
  - Missing setters.
  - No method implementation
- Contextual: 1-0/9 =1
  - 9 classes: User, UserManager, Wallet, WalletManager, Asset, AssetManager, SmartContract, SmartContractManager, Transaction,

P9: Develop tests including unit tests, integration tests, and system tests for the implementation.

- Notational: 1-(1*0.8)/3 = 0.733
  - Unit tests (1), integration tests (1), system tests (1)
  - No implementation for system tests.
- Contextual: 1-(5+5*0.5)/15 = 0.5
  - 5 Unit Tests
  - 5 Integration Tests
  - 5 System Tests
  - Integration tests involving only one class.
  - System tests involving only two classes.