Principle of Distributed Ledgers

Lecture 3: Smart Contract Development

Lewis Gudgeon, **Daniel Perez**, Paul Pritz, Sam Werner

February 3, 2023

Imperial College London

Development flow for smart contracts

1. Write high-level code
2. Test the code (using testing suite of choice, e.g. Hardhat, Brownie, Foundry)
3. Optimise the code for gas efficiency
4. Compile the contract into Bytecode
5. Send a transaction to deploy the contract
6. Interact with the contract by sending transactions to the generated address

Development flow for smart contracts

1. **Write high-level code**
2. **Test the code (using testing suite of choice, e.g. Hardhat, Brownie, Foundry)**
3. Optimise the code for gas efficiency
4. Compile the contract into Bytecode
5. Send a transaction to deploy the contract
6. Interact with the contract by sending transactions to the generated address

Interface definition

An interface contains the function signatures of a contract, but not the implementation. It is used to define the contract's public API.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  interface IMyContract {
5      function totalSupply() external view returns (uint256);
6
7      function name() external view returns (string memory);
8
9      function mint(uint256 amount) external;
10 }
```

Library definition

A library contains function definitions that can be used by other contracts.

- Libraries with one or more public functions need to be deployed as a contract itself (dynamic linking).
- Libraries with only internal functions are embedded in contracts (static linking).

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3 library MyLibrary {
4     function computeNewSupply(uint256 oldSupply, uint256 amountMinted)
          internal view returns (uint256) {
5         return oldSupply * 11 / 10 + amountMinted;
6     }
7 }
```

Contract definition

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3  import "./IMyContract.sol";
4  import "./MyLibrary.sol";
5  contract MyContract is IMyContract {
6      uint256 public totalSupply;
7      string public name;
8      constructor (string memory name_, uint256 initialSupply_) {
9          name = name_;
10         totalSupply = initialSupply_;
11     }
12     function mint(uint256 amount) external {
13         totalSupply = MyLibrary.computeNewSupply(totalSupply, amount);
14     }
15 }
```

Development tools

Hardhat Uses JavaScript/TypeScript for testing/deployment. Likely the most widely used. `https://hardhat.org/`

Brownie Uses Python for testing/deployment. Easy to use but slightly under-maintained. `https://eth-brownie.readthedocs.io/`

Foundry Uses Solidity for testing/deployment paired with CLI commands. Newer but much faster than the others. `https://book.getfoundry.sh/`

Development tools

Hardhat Uses JavaScript/TypeScript for testing/deployment. Likely the most widely used. `https://hardhat.org/`

Brownie Uses Python for testing/deployment. Easy to use but slightly under-maintained. `https://eth-brownie.readthedocs.io/`

Foundry Uses Solidity for testing/deployment paired with CLI commands. Newer but much faster than the others. `https://book.getfoundry.sh/`

We will use Foundry for this course

Project structure

- `foundry.toml` - Configuration file for Foundry
- `remappings.txt` - File containing the remappings for the project
- `src` - Directory containing the source code. Interfaces and libraries are often in their own directory
- `test` - Directory containing the test code

```
.
|-- foundry.toml
|-- remappings.txt
|-- src
|   |-- ERC20.sol
|   |-- ERC721.sol
|   |-- interfaces
|   |   |-- IERC20.sol
|   |   `-- IERC721.sol
|   `-- libraries
|       `-- StringUtils.sol
`-- test
    |-- ERC20.t.sol
    `-- ERC721.t.sol
```

A word about ERC

- ERC stands for Ethereum Request for Comment. It is the equivalent of an RFC in the Ethereum community
- ERCs are used to add/modify Ethereum features or to standardize the implementation of certain features in Ethereum
- ERCs have later been renamed to EIPs (Ethereum Improvement Proposals) but the name ERC is still widely used

ERC-20 tokens

- ERC-20 is a standard for fungible tokens
- Fungible tokens are tokens that are interchangeable with each other, i.e. they have the same value
- ERC-20 tokens can be used to represent currencies, shares, etc.
- ERC-20 tokens have functionalities to check balances, transfer tokens, approve other accounts to transfer tokens, etc.

Fixed point arithmetic

- A major issue when working with ERC-20 tokens (and Ether) is the fact that they are represented as integers
- This means that by default, we cannot represent fractions of tokens
- To work around this, tokens are often represented as integers with a fixed number of decimals, e.g.
  - 1 token is represented as 10^{18}
  - this means that 0.1 token would be 10^{17}
- Computations need to take this into account to keep the scaling consistent

ERC-721 token

- ERC-721 is a standard for non-fungible tokens (NFT)
- NFTs are tokens that are not interchangeable with each other, i.e. they represent something unique
- A typical use case for NFTs is to represent ownership of a digital asset
- NFTs usually have an associated URI pointing to the metadata of the digital asset
- The associated digital asset and its metadata are usually stored on another storage system, e.g. IPFS

Useful resources

- Solidity documentation (`https://docs.soliditylang.org/`)
- Foundry documentation (`https://book.getfoundry.sh/`)
- ERC-20 standard (`https://eips.ethereum.org/EIPS/eip-20`)
- ERC-721 standard (`https://eips.ethereum.org/EIPS/eip-721`)