FIDELITY CENTER for APPLIED TECHNOLOGY®

AN OVERVIEW OF SOLIDITY DEVELOPMENT FRAMEWORKS

By: Chris Helms & Chris McGahon

May, 2023

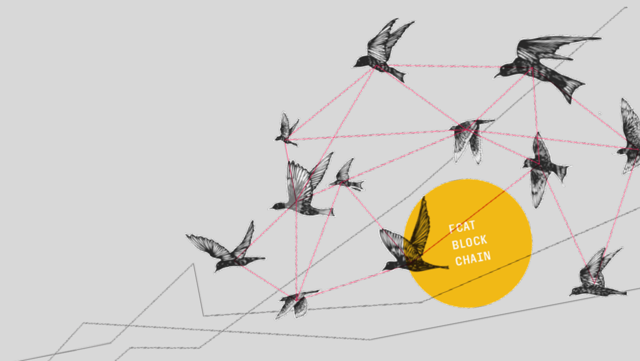
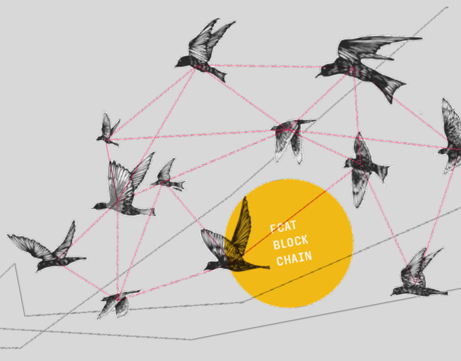


TABLE OF CONTENTS



Executive Summary

Since its introduction in 2014, solidity development has been a rapidly growing and exciting topic within the world of blockchains and more specifically Ethereum. During its growth, there have been many different tools & frameworks that support smart contract development and testing. But with many different choices and paths to go down, what is the best way for developers to traverse the ever growing & changing options? Here we explore modern frameworks and their benefits when building smart contracts.

What is a Framework?

With Solidity development growing in popularity, engineers require additional support from tools for building smart contracts safely and efficiently. While online editors are simple to launch and useful for experimentation, development frameworks are often preferred by those who want more control over how their environment runs and behaves. Because smart contracts are a newer technology, the best approaches for features and tools within these frameworks are still being determined. Our goal is to explore the most popular Solidity development frameworks and explain their features, the key differences between them, our insights, discuss market trends, and provide a starting point for those interested in learning more about these frameworks.

Modern Solidity Frameworks

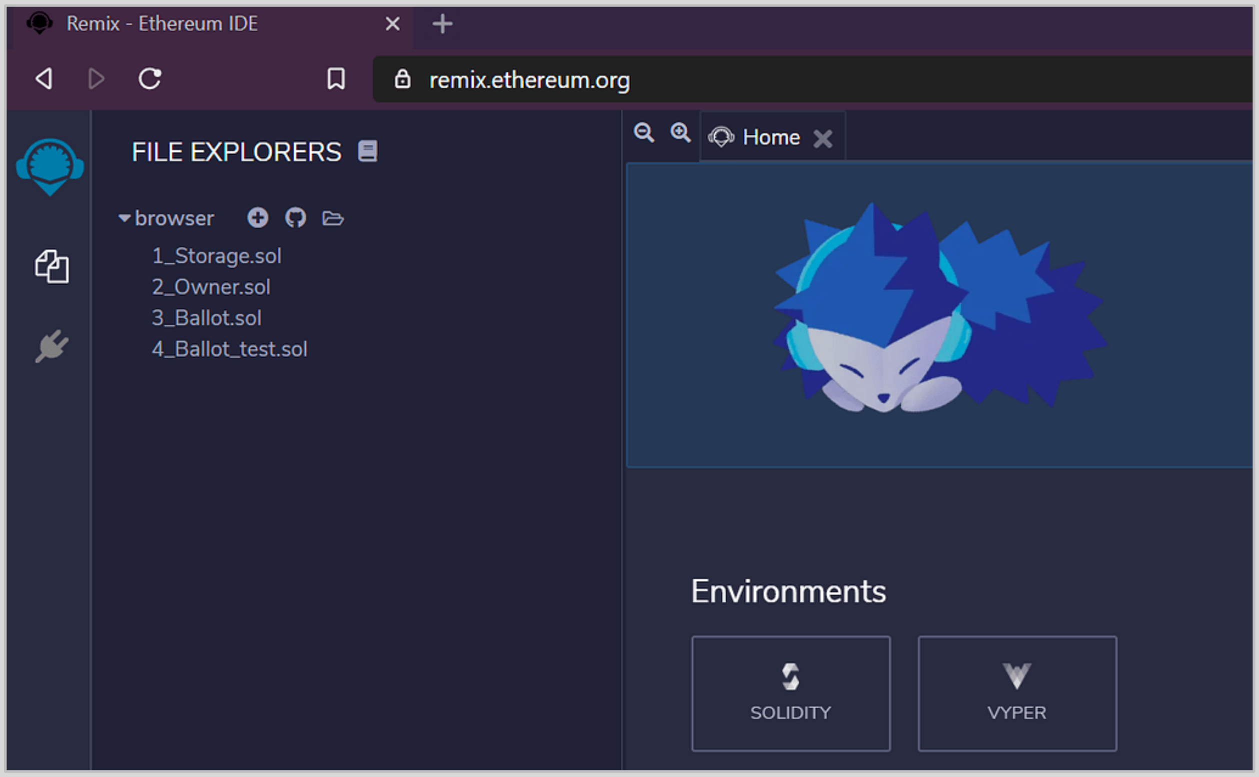


Figure 1: Remix, a browser-based IDE running in browser.

Remix

Remix is the most common starting point into the foray of writing smart contracts (Fig. 1). As a browser based integrated development environment (IDE), it requires little setup and is readily available for anyone interested to use. Remix contains a plethora of different tools for development, including a basic workspace, compiler, debugger, and can be configured to deploy to most major remote procedure call (RPC) services. Because of its simplicity, using

Remix only requires a basic understanding of blockchain concepts and some experience with Solidity.

While the online IDE is the most popular form of Remix, its ecosystem is rapidly expanding. Remix also offers a desktop IDE and Visual Studio Code plugin that provide additional functionality and a more secure approach of storing files locally on developer's machines.

Additional information on Remix: https://remix.ethereum.org/

Truffle

Truffle is a Node-based framework created by ConsenSys in 2016. It comes packaged with tools for developing, compiling, JavaScript-based testing, and deploying Ethereum smart contracts.

Truffle requires that users define a project with a specific organizational structure and configuration file so it can better aid in development efforts. As part of these efforts, Truffle utilizes migrations to manage contract versions. Migrations are a concept used across frameworks and suites that compare current code to deployed code and ensure that the latest version is deployed prior to testing. This approach means the developer does not need to consider which contracts should be redeployed after code changes are made. The Truffle suite also has a proprietary development network called Ganache. When Truffle projects are deployed to Ganache, they have speed and debugging benefits, as well as additional contract insight and interaction through the Ganache GUI. Overall, the Truffle ecosystem contains many tools that were designed to enhance the development experience when used together.

Additional information on Truffle: https://trufflesuite.com/

HardHat

Hardhat is a Node based framework created by the Nomic Foundation. The framework shares many features with Truffle including development support, JavaScript testing, and deploy of contracts. Hardhat has been growing in popularity, boasting a wide range of open-source plugins, and increasing download counts through NPM [1]. Plugins include gas analyzers, unit test coverage reports, and additional capabilities for unit testing.

Unlike Truffle, deployed contract versions in Hardhat are managed by the developer. This requires developers to handle contract deployments themselves, usually through a JavaScript file combined with various Hardhat packages. Hardhat projects are highly configurable and allow developers to hook into various parts of the project lifecycle for more granular control over the framework's behavior.

Additional information on Hardhat: https://hardhat.org/

Foundry

Foundry is a framework that has recently been gaining popularity with smart contract developers for its rapidly growing toolset and speed. Foundry's features are accessed entirely though a command line interface (CLI) that is made of three parts:

- **Forge -** contract compilation, deploys, and testing.
- **Cast -** transaction creation, blockchain interaction, data conversions.
- **Anvil -** local network for testing and debugging contracts.

Foundry is written in a programming language called Rust, so its approach to development differs from the previously mentioned Node-based Frameworks. Dependencies are installed using Git submodules instead of NPM packages. Unit tests are written using Solidity through extension of the DSTest library, an approach taken to ensure all smart contract related development utilizes the same language.

Foundry focuses on speed, with compilation and testing taking notably shorter time to run than its competitors. Fig 2 shows the difference in Foundry and Hardhat compilation times using various caching strategies.

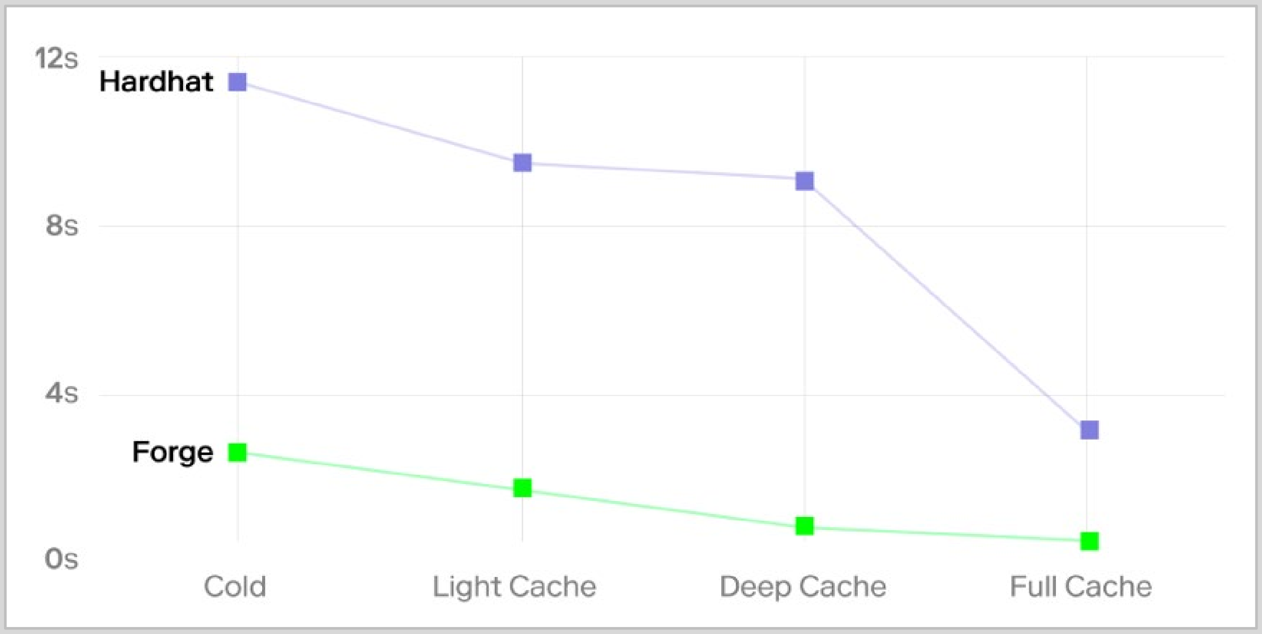


Figure 2: Forge vs Hardhat compilation times [2]

Additional information on Foundry: https://book.getfoundry.sh/

Market Trends & Popularity

Developers are constantly evaluating tools and features to determine what best fits their needs in the rapidly changing blockchain space. For early adopters of Solidity, Truffle was the standard framework for smart contract development. Because of its rich feature set and large variety of plugins, many projects have been transitioning to Hardhat as of publication. The change in framework preference can be seen in the below NPM weekly download graphs, with Truffle's downloads decreasing (fig 3) and Hardhat's steadily increasing (fig 4). Finally, while Foundry is a developing framework and is still building a full set of capabilities, many recent projects prefer it for its compilation and testing speed.

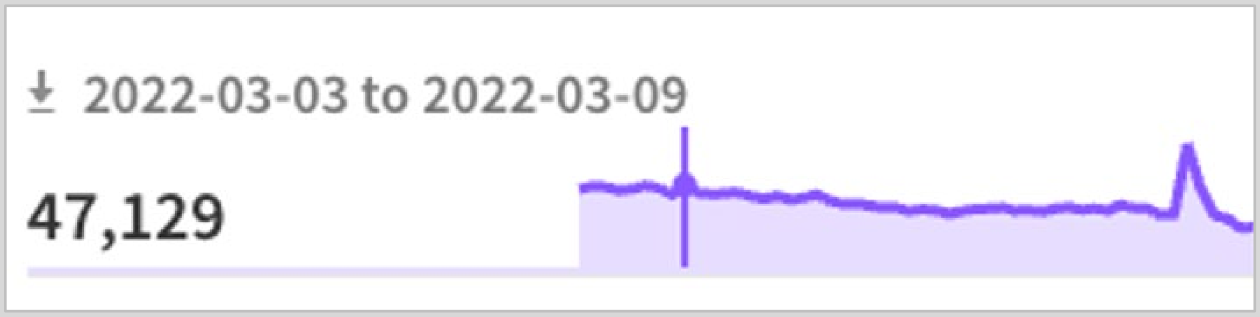


Figure 3: Truffle's weekly NPM downloads [3]

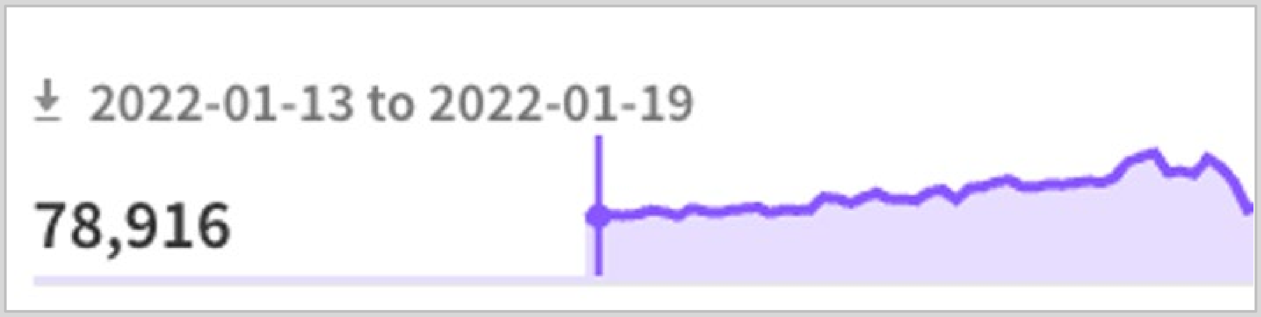


Figure 4: Hardhat's weekly NPM downloads [4]

Framework Features

With Solidity development growing in popularity, engineers require additional support from tools for building smart contracts safely and efficiently. While online editors are simple to launch and useful for experimentation, development frameworks are often preferred by those who want more control over how their environment runs and behaves. Because smart contracts are a newer technology, the best approaches for features and tools within these frameworks are still being determined. Our goal is to explore the most popular Solidity development frameworks and explain their features, the key differences between them, our insights, discuss market trends, and provide a starting point for those interested in learning more about these frameworks.

Configurations

Most modern frameworks have settings that control development environment behavior. These settings include options for compilers, networks, and accounts.

A configuration file is used to define a baseline for the settings and can be checked into source control to maintain consistency for all developers working on the project. The following are some options commonly defined in project configurations:

- **Compiler Version / Solc -** Framework configurations, along with the project's Solidity code, determine which compiler version to use and how to obtain it. Common options include downloading and running the compiler binary directly or performing compilations using a docker container.

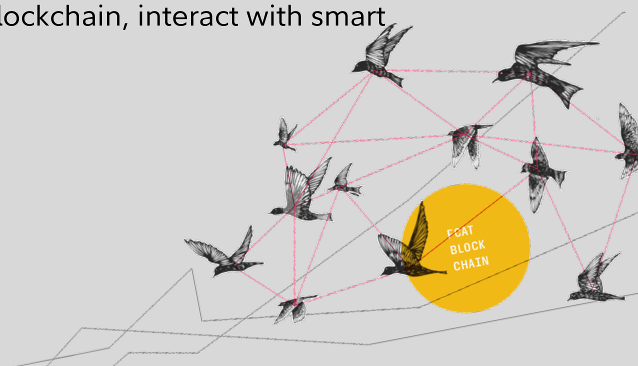
- **Networks -** Configurations additionally define a set JSON-RPC or web socket endpoints that represent various Ethereum networks. These networks define separate environments and can be used for various parts of the development lifecycle, including development testing or production deploys. In most frameworks, a specific network can be referenced as a target when using scripts or CLI commands.

- **Accounts -** Any Ethereum accounts used when developing or deploying to production can also be defined in the project configuration. This allows developers to easily access and re-use the set of accounts they require when interacting with an application. Accounts are typically defined using a mnemonic or private keys directly, and can be referenced from scripts, tests, and when configuring networks. When using a mnemonic, most frameworks contain easy to use libraries for generating new accounts and transaction signing.

Development Networks

When writing smart contracts, it is important to test features and changes before deploying them to a public network. Solidity frameworks are commonly shipped with additional software known as "development networks". The most popular options include Hardhat Network, Ganache, and Anvil.

Development networks allow developers to run an Ethereum Network node on their local machine. These nodes act as a full-fledged blockchain and come packaged with a series of additional features that are useful during Solidity development. Nodes also expose both JSON-RCP and WebSocket endpoints that mimic public network node behavior. Developers can use these endpoints to retrieve information about the blockchain, interact with smart contracts, transfer Ether, etc.



Some of the most valuable tools these networks provide pertain to debugging and environment control. Development nodes expose additional functionality to manipulate blockchain properties for testing purposes. These features include manual block-time definition, swapping between manual and auto-mining strategies, and mimicking transactions sent from a chosen address. Because these capabilities often result in invalid blockchain states, they are limited to development environments and not available on real world nodes.

Development networks also support network forking. Forking allows developers to duplicate an existing blockchain's state, including contracts, data, and Ethereum balances for local use. Forking is useful for examining contracts on external networks and testing planned transactions against them. Since forking captures the full state of a target chain, testing a contract on a fork may reveal problems or bugs prior to deploying to its real-world counterpart.

Unit Testing

Like Web2, the blockchain development lifecycle uses unit testing to validate the basic behavior of code. Truffle and Hardhat unit tests are written in JavaScript and utilize the popular testing libraries Mocha and Chai. Foundry unit tests are written in Solidity, with testing classes based on the DSTest library. Each approach to testing has pros and cons, with JavaScript based tests being generally more familiar to developers and Solidity based tests more naturally integrating with the smart contracts being tested.

During unit testing, developers require ways to manipulate Ethereum properties like account balances, gas prices, and block-time. Each framework uses a separate library to provide access to these controls and other useful testing tools. Using these libraries, a typical unit testing flow would include account preparation, contract deploy and setup, running test-specific interaction with the contract, and validation of the blockchain state afterwards. Some libraries offer additional features for improved testing, such as handles to previous blockchain states for faster testing and additional logging capabilities.

Scripts

While most frameworks expose a CLI that can be used for singular actions, developers often need to run many commands back-to-back or handle more complex requirements. Scripts are a general term for code that performs actions against a blockchain. This may include deploying and interacting with contracts, transferring ETH, retrieving account details, etc.

Scripts rely on underlying libraries to interface with the desired blockchain. Web3js is used by Truffle, while Ethers is preferred by Hardhat. While these libraries are separate, they provide similar capabilities for contract interaction and testing. Because Foundry is based in Rust, its approach is different from its node-based counterparts. Foundry scripts are mainly written in Bash and use the Cast CLI for all blockchain related needs.

Scripts can be written to handle a series of interactions with a blockchain. Most frameworks additionally support a variety of plugins that define additional features. An example is the TypeChain plugin, which generates helper classes for Typescript code based on a project's smart contracts. Scripts generally reference a specific configuration when they are run that may include specific network and account settings. This allows scripts to be written generically and run against multiple environments, contracts, or accounts depending on development requirements.

Plugins

Developers may determine an additional need or functionality for their project that is not included out of the box with their framework. Plugins are installed alongside frameworks and provide expanded features and tools. The following section covers common plugins used for developing and testing smart contracts.

TypeChain

TypeChain is designed to help with contract interaction when using Typescript. By generating types based on smart contract ABIs, developers are given easier access to contract functions and state variables. This improves usability when writing tests and scripts in Typescript. TypeChain contains support for the existing Node based frameworks, along with multiple settings for various project configurations.

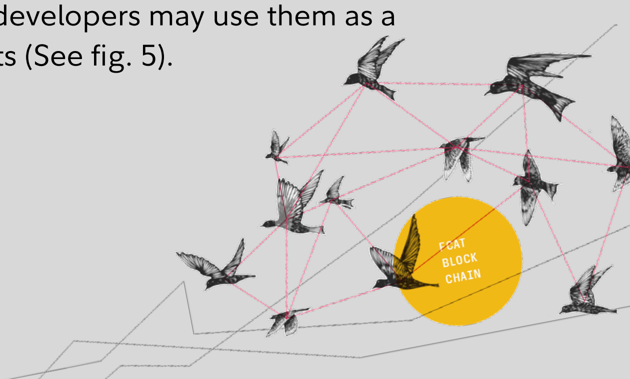
Additional information on TypeChain: https://github.com/dethcrypto/TypeChain

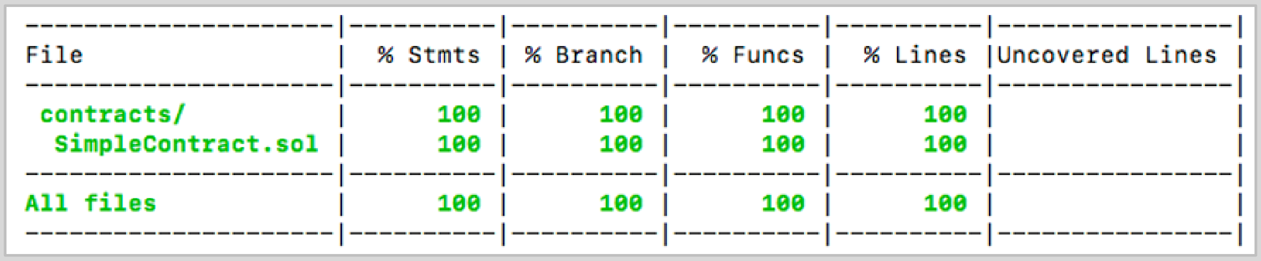
Code Coverage

Code coverage is a metric used software developers use to determine which lines, conditionals, and functions are covered while testing. A coverage report is printed with stats broken down by file and function. Plugins may also show graphical indications of coverage by highlighting tested lines of code within the IDE.

While Solidity frameworks normally ship with unit testing capabilities, not all of them include features for outputting test coverage reports. Truffle and Hardhat can be modified to perform this behavior by installing the solidity-coverage package [5], which creates coverage reports and can be further configured to display a graphical overlay within Visual Studio Code. Alternatively, Foundry natively includes unit testing coverage as part of the Forge CLI which can generate reports in multiple formats but does not include graphical capabilities for IDEs.

Because development tools for smart contracts are still within their infancy, accuracy and behavior is not yet guaranteed for coverage reports. Complex contract designs like upgradeability can cause inconsistent behavior in the above tools. Since these features are still being improved to work with more advanced contracts, developers may use them as a guide for unit testing quality and validate the resulting reports (See fig. 5).





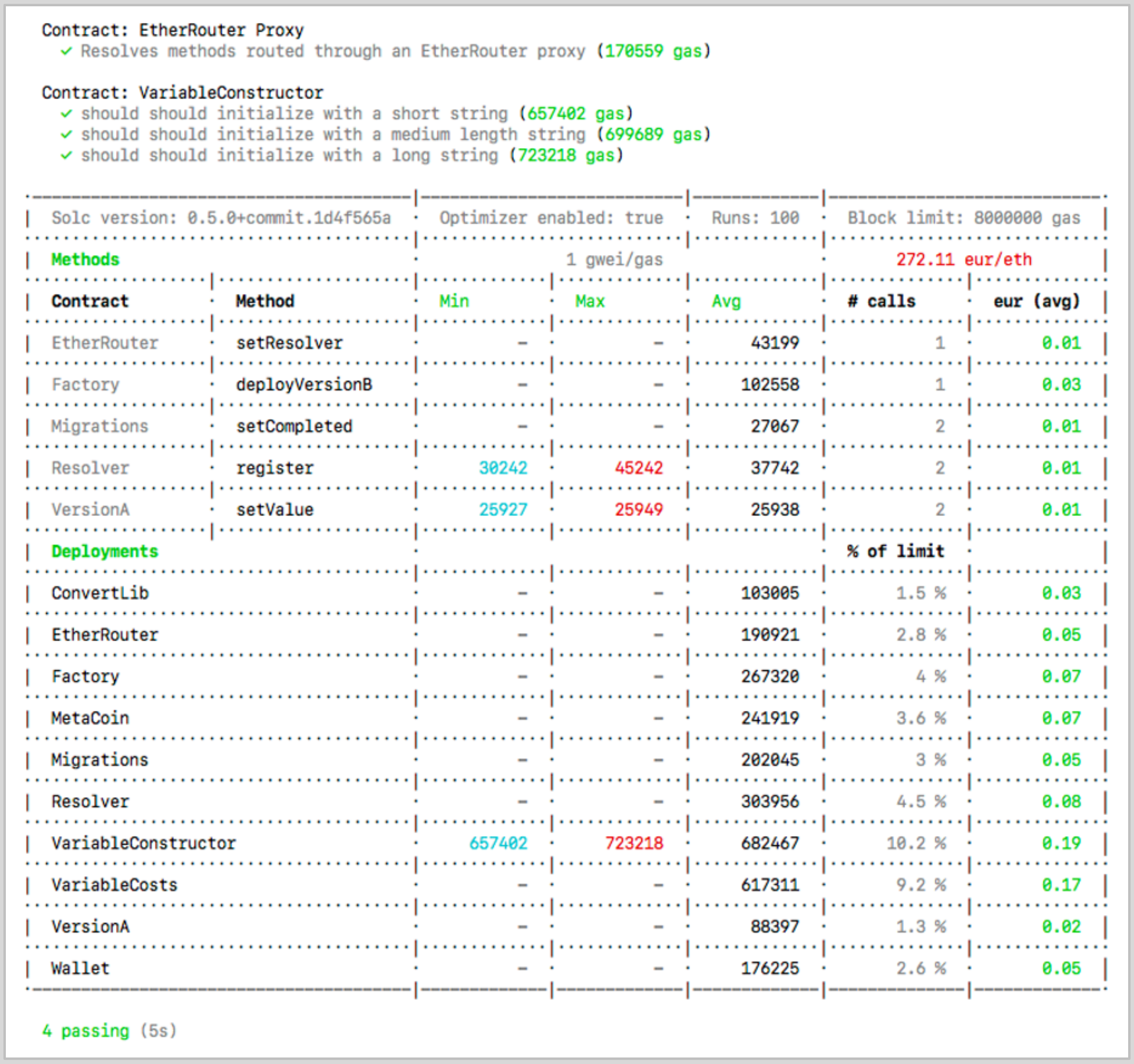
```
----------------------|----------|----------|----------|----------|----------------|
File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------|----------|----------|----------|----------|----------------|
 contracts/           |      100 |      100 |      100 |      100 |                |
  SimpleContract.sol  |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
All files             |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
```

Figure 5: Solidity Coverage [6]

Gas Analysis

A key difference between developing smart contracts and traditional programs is the concept of gas, the resource used to carry out operations on the blockchain. Because gas acts as a digital commodity, it is important to ensure smart contracts are optimized and use it efficiently. A useful tool in determining this metric is a gas analyzer, which creates a report on the gas used by a smart contract. These reports can be used to determine if a contract is using gas optimally or to track changes in efficiency when developing contract code.

Out of the box, Remix can be configured to automatically show the gas analysis of smart contracts. Foundry also ships with native gas analysis support using the Forge CLI. Truffle and Hardhat require additional plugins to expose this feature, with the most used being eth-gas-reporter [7] and hardhat-gas-reporter [8]. By installing and configuring these packages within a project, a gas analysis can be printed to the terminal when each of the unit tests are run. An example of this report is shown for a Hardhat project in Fig 6.

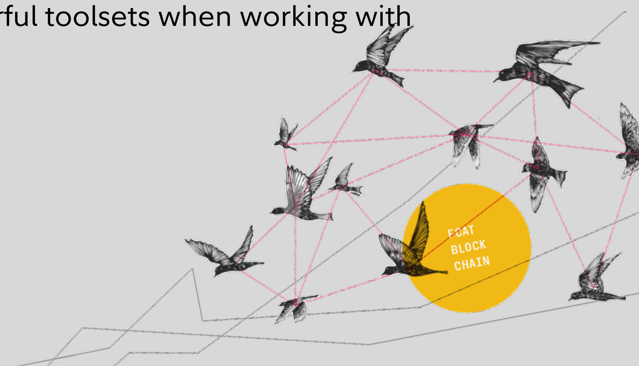


```
  Contract: EtherRouter Proxy
    ✓ Resolves methods routed through an EtherRouter proxy (170559 gas)

  Contract: VariableConstructor
    ✓ should should initialize with a short string (657402 gas)
    ✓ should should initialize with a medium length string (699689 gas)
    ✓ should should initialize with a long string (723218 gas)

·----------------------------------------|---------------------------|-------------|----------------------------·
|  Solc version: 0.5.0+commit.1d4f565a   ·  Optimizer enabled: true  ·  Runs: 100  ·  Block limit: 8000000 gas  │
·········································|···························|·············|·····························
|  Methods                               ·          1 gwei/gas       ·       272.11 eur/eth                      │
··················|······················|·············|·············|·············|··············|··············
|  Contract       ·  Method              ·  Min        ·  Max        ·  Avg        ·  # calls     ·  eur (avg)  │
··················|······················|·············|·············|·············|··············|··············
|  EtherRouter    ·  setResolver         ·          -  ·          -  ·      43199  ·           1  ·       0.01  │
··················|······················|·············|·············|·············|··············|··············
|  Factory        ·  deployVersionB      ·          -  ·          -  ·     102558  ·           1  ·       0.03  │
··················|······················|·············|·············|·············|··············|··············
|  Migrations     ·  setCompleted        ·          -  ·          -  ·      27067  ·           2  ·       0.01  │
··················|······················|·············|·············|·············|··············|··············
|  Resolver       ·  register            ·      30242  ·      45242  ·      37742  ·           2  ·       0.01  │
··················|······················|·············|·············|·············|··············|··············
|  VersionA       ·  setValue            ·      25927  ·      25949  ·      25938  ·           2  ·       0.01  │
··················|······················|·············|·············|·············|··············|··············
|  Deployments                           ·                                         ·  % of limit  ·             │
·········································|·············|·············|·············|··············|··············
|  ConvertLib                            ·          -  ·          -  ·     103005  ·       1.5 %  ·       0.03  │
·········································|·············|·············|·············|··············|··············
|  EtherRouter                           ·          -  ·          -  ·     190921  ·       2.8 %  ·       0.05  │
·········································|·············|·············|·············|··············|··············
|  Factory                               ·          -  ·          -  ·     267320  ·         4 %  ·       0.07  │
·········································|·············|·············|·············|··············|··············
|  MetaCoin                              ·          -  ·          -  ·     241919  ·       3.6 %  ·       0.07  │
·········································|·············|·············|·············|··············|··············
|  Migrations                            ·          -  ·          -  ·     202045  ·         3 %  ·       0.05  │
·········································|·············|·············|·············|··············|··············
|  Resolver                              ·          -  ·          -  ·     303956  ·       4.5 %  ·       0.08  │
·········································|·············|·············|·············|··············|··············
|  VariableConstructor                   ·     657402  ·     723218  ·     682467  ·      10.2 %  ·       0.19  │
·········································|·············|·············|·············|··············|··············
|  VariableCosts                         ·          -  ·          -  ·     617311  ·       9.2 %  ·       0.17  │
·········································|·············|·············|·············|··············|··············
|  VersionA                              ·          -  ·          -  ·      88397  ·       1.3 %  ·       0.02  │
·········································|·············|·············|·············|··············|··············
|  Wallet                                ·          -  ·          -  ·     176225  ·       2.6 %  ·       0.05  │
·----------------------------------------|-------------|-------------|-------------|--------------|-------------·

  4 passing (5s)
```

Figure 6: Gas report sample [9]

Conclusion

This article provides an overview of Solidity frameworks and their features. As smart contract development grows, so do the related features and tools. The constantly evolving nature of the field is shown with user trends amongst Remix, Truffle, Hardhat, Foundry, and their open-source plugins. Understanding new features and their uses sharpens developer skills and their understanding of the blockchain ecosystem. Using these frameworks, developers of any skill level can begin utilizing best practices and powerful toolsets when working with Solidity.



	Truffle	**Hardhat**	**Foundry**
Language/Framework	Node	Node	Rust
Development Network	Ganache	Hardhat Network	Anvil
Unit Testing	JavaScript	JavaScript	Solidity
Scripts	JavaScript	JavaScript	Bash
Code Coverage Support	Moderate	Full	Moderate
Gas Analysis Support	Full	Full	Full

Figure 7: Framework feature comparison.

6. REFERENCES

1. NPM. (2022). NPM.
https://www.npmjs.com/

2. Foundry-Rs. (2023). Foundry-rs/foundry: Foundry is a blazing fast, portable and Modular Toolkit for Ethereum application development written in rust. GitHub.
https://github.com/foundry-rs/foundry

3. Truffle. (2022). NPM.
https://www.npmjs.com/package/truffle

4. Hardhat. (2022). NPM.
https://www.npmjs.com/package/hardhat

5. Solidity-coverage. (2022). NPM.
https://www.npmjs.com/package/solidity-coverage

6. Digitalime. (2018). Solidity unit tests & code coverage with etherlime. Medium.
https://medium.com/limechain/solidity-unit-tests-code-coverage-with-etherlime-9e2aa8da516a

7. Eth-gas-reporter. NPM. (2022).
https://www.npmjs.com/package/eth-gas-reporter

8. Hardhat-gas-reporter. NPM. (2022).
https://www.npmjs.com/package/hardhat-gas-reporter

9. Cgewecke. (2022). Cgewecke/ETH-gas-reporter: Gas usage per unit test. average gas usage per method. A mocha reporter. GitHub.
https://github.com/cgewecke/eth-gas-reporter