On Ink Integration Tests

Milestone Report

September, 2023

Milestone 1

Timeline

Started on 2023-08-28, ending on 2023-09-08.

Total duration: 2 weeks.

What was defined

1. **Documentation.** Create a comprehensive report that compares the functionalities of integration tests and E2E (End-to-End) tests. The report's focus is to identify what can be accomplished in E2E tests but not currently in Integration tests, as well as any inconsistencies. If applicable, we will provide suggestions that are not covered by either test type.
2. **Article.** We will prepare a summary report and publish it on our blog https://blog.coinfabrik.com/ .
3. **Analyze**. Study and compare Integration and E2E tests in ink!.
4. **Evaluate.** Assign a complexity level to each finding based on the difficulty of implementing the missing or enhanced functionality.
5. **Estimate.** Indicate an order of priority under which the missing functionalities shall be developed during the next milestone, where the functionalities are effectively implemented for integration tests.

What we did

1. **Documentation.** Check the report included herein as Appendix 1.
2. **Article.** Check our summary report published in our blog.
3. **Analyze.** Included within the Documentation of Appendix 1.
4. **Evaluate.** See complexity level for the implementation of each finding in Reference 1: Findings and Estimations, in Results section of Appendix 1, where we also provide a guideline for its implementation.
5. **Estimate.** See the implementation order of priority of the analyzed issues in Reference 1: Findings and Estimations, in Results section of Appendix 1. This order was established considering the feasibility of the implementation, as well as the dependency order among features.

Appendix 1: Comparing functionalities of Integration and E2E tests in ink!

Background

Integration tests run significantly faster than E2E tests. If a full range of functionalities were provided, it could reduce testing and QA times. Ideally, each test in E2E should have its counterpart in Integration.

We have discovered that Integration tests for ink! contracts lack some of the functionalities found in E2E testing.

These gaps in functionality came to our attention during the development of fuzzing detectors for our vulnerability analyzer, Scout. During this work, we identified 2 functions with implementation differences, default_accounts() and set_contract_storage(). We also observed 3 functions which were unimplemented for integration tests. This initial identification was observed in our grant application form for the grant #1875.

In this report, we provide a complete list of unimplemented features, establish a methodology for finding implementation differences, define an exploration roadmap for unevaluated issues and an implementation roadmap for those we were able to analyze in depth during the assigned time of 2 weeks.

Scope

In order to determine the universe of analysis, we reviewed the repositories listed in Table 1.

Table 1: Reviewed repositories

Repository	Reviewed Commit
https://github.com/paritytech/ink	c2af39883aab48c71dc09dac5d06583f2e84dc54
https://github.com/paritytech/substrate	28e906dffcaa91e85f59aff628d953ebeb036ae2
https://github.com/paritytech/polkadot	52209dcfe546ff39cc031b92d64e787e7e8264d4

Upon analysis, we narrowed down our search to the repositories where the testing functionalities for Integration and E2E tests were implemented (see Table 2).

Table 2: Implementation repositories

Test Type	Implementation Repositories	Reviewed Commit
Integration	https://github.com/paritytech/ink/blob/master/crates/env/src/engine/off_chain/test_api.rs (only for default_accounts()) https://github.com/paritytech/ink/blob/master/crates/env/src/engine/off_chain/impls.rs https://github.com/paritytech/ink/blob/master/crates/engine/src/ext.rs	c2af39883aab48c71dc09dac5d06583f2e84dc54
E2E	https://github.com/paritytech/ink/blob/master/crates/env/src/engine/on_chain/impls.rs	c2af39883aab48c71dc09dac5d06583f2e84dc54
	https://github.com/paritytech/substrate/blob/master/frame/contracts/src/wasm/runtime.rs	28e906dffcaa91e85f59aff628d953ebeb036ae2

Reviewing the implementation files above, we decided to focus our analysis on the functions exposed for their usage in integration and e2e tests in the `env_access.rs` file.

Table 3: Exposed Functions

Test Type	Exposed Functions	Reviewed Commit
Integration	https://github.com/paritytech/ink/blob/master/crates/ink/src/env_access.rs	c2af39883aab48c71dc09dac5d06583f2e84dc54
E2E		

A comprehensive list of all functions to analyze is provided in Attachment 1: Universe of Functions.

Methodology

1. We established the universe of functions to analyze listing the functions exposed for their usage in integration and e2e tests in the `env_access.rs`, and also considering default_accounts() and set_contract_storage(), which we identified while performing tests for Scout. See this list in Attachment 1: Universe of Functions. Our focus was

put on reviewing clearly unimplemented functions or functions with obvious implementation differences between integration and E2E tests.

2. We performed a complete search of the ink!, substrate and polkadot repositories mentioned in Table 1 above looking for unimplemented functions, that is, functions with the unimplemented!() macro within its definition but no real implementation. Of a total of 73 functions using the unimplemented!() macro, 13 functions correspond to implementations for integration tests and none for implementations of E2E tests. Of these 13, only 9 were accessible for their usage in integration or e2e tests, and included in our Attachment 1: Universe of Functions.

3. For an initial analysis list, we considered these 9 unimplemented functions for integration tests and 2 additional functions with implementation differences between Integration and E2E tests found during the development of Scout: default_accounts() and set_contract_storage(). This way, we constructed an initial analysis list of 11 functions. See Attachment 2: Initial Analysis List.

4. During the two weeks assigned for this grant, we analyzed in depth and documented our findings for functions issue number 1 through 11 of Attachment 2: Initial Analysis List. This analysis consisted of:

    a. Creating a smart contract example to test the function in Integration tests and E2E tests.

    b. Writing at least one test case per function per environment (*). We chose to label the tests as failing/not-passing when the implementation is incorrect or absent in the Integration environment. These tests will run properly once the implementation is completed or fixed.

    c. Comparing the results of running each test. Showing that the function works as intended in E2E test but missing development or performing in an unintended way for integration tests, or showing the suspected implementation difference.

    d. Evaluating the complexity level of each finding, that is, providing an implementation guideline and implementation time estimate.

    (*) Ideally, each test case in E2E should have its counterpart in Integration. This implies that the set of tests per function is unique and should be replicated in both environments. This idea will be reviewed in the next milestone, as achieving such a level of thoroughness is not planned for this instance. Therefore, the acceptance criteria for now has been set to ensure that there is at least one test case per function.

5. Based on our findings, we determined an order of priority for implementation, considering feasibility, implementation time and implementation dependencies between the analyzed functions.

Results

From our analysis, we observe that only integration test functions use the `unimplemented!()` macro, all E2E functions for testing show implementations.

We have successfully demonstrated that the E2E counterparts of the 9 unimplemented functions in the integration environment worked as expected. This validation was carried out by employing at least one smart contract example per analyzed function, and one test case per test environment.

Regarding the 2 functions with known differences, default_accounts() and set_contract_storage(), the smart contract examples and associated tests serve as documentation for the identified disparities. All test case examples can be found in our publicly available analysis repository.

We also identified, although this is not associated with a missing function implementation, the need to provide true accounts for integration tests. That is, accounts that behave like they do in e2e tests. This is necessary for the implementation of the function `instantiate_contract()`, `instantiate_contract()`, `invoke_contract()`, `code_hash()` and `own_code_hash()`.

All in all, out of the 11 analyzed functions, we deemed as feasible and estimated the implementation of 9 of them, leaving out as impractical the implementation of `gas_left()` and `call_runtime()`.

We summarize below our findings and estimations. On top of the Implementation Time Estimate, each implementation should be complemented with additional QA Time (50%* FTE_IMPLEMENTATION) and Documentation Time (25% * FTE_IMPLEMENTATION).

Reference 1: Findings and Estimations

Proposed Implementation Order	#1
Function	`default_accounts`
Issue Number	1
Status	Implementation Difference
Issue Description	Alice and Bob's addresses should match across integration and E2E tests.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/default-accounts

Implementation Idea	This is relatively trivial. It can be implemented in a single day. For backwards compatibility django and frank should retain their current names, and dave and ferdie added as their respective aliases.
Complexity Level	Feasible
Implementation Time Estimate	1 day [FTE=1]

Proposed Implementation Order	#2
Function	`set_contract_storage`
Issue Number	2
Status	Missing limitation on Integration Testing
Issue Description	The storage in the integration environment should have the same limitations as in the blockchain environment (end-to-end).
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/set-contract-storage
Implementation Idea	This feature is practically complete, it's just missing a size check. Implementing this check should be fairly trivial. The best place to do it would probably be in the set_storage() implementation in ink/crates/engine/src/ext.rs, probably by simply panic!()ing (as there's no way to communicate an error).
Complexity Level	Feasible
Implementation Time Estimate	1 day [FTE=1]

Proposed Implementation Order	#3
Function	`instantiate_contract()`
Issue Number	#7

Status	Missing Function Implementation on Integration Testing
Issue Description	No implementation of instantiate contract for integration tests. This prevents invoking one contract from another.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/blob/main/test-cases/instantiate-contract

Implementation Idea	The implementation of `invoke_contract()` is somewhat tied to that of `code_hash and own_code_hash()`. This is why we repeat parts of the implementation idea for the three of them. Nevertheless, implementation times should decrease, as the implementation of the second and third functions will reuse solutions developed for the first.
	
	This function requires accounts and contract instantiation to be implemented. After that's done, there are two main methods to implement this functionality:
	
	1. Hash some property of the specified contract. For example, the address of the constructor.
	2. Implement the code hash equivalently to how it's done on E2E.
	
	Option #1 is fairly easy to implement. However, it can only support comparing hashes returned by either `own_code_hash()` or `code_hash()`, not if the caller specifies a hash directly. E.g.
	
	if code_hash(foo) == "0xf00..."{
	}
	
	Option #2 supports both use cases, but it's more complex to implement and also slows down integration tests by requiring the full WASM compilation of the contract.
	
	There are some things to work out, such as how to call the required function, that add some uncertainty to the estimate.

Complexity Level	Complex, yet feasible
Implementation Time Estimate	10 days [FTE=1] (for Option 1, suggested) 15 days[FTE=1] (for Option 2)

Proposed Implementation Order	#4

Function	code_hash
Issue Number	9
Status	Missing Function Implementation on Integration Testing
Issue Description	`code_hash´ retrieves the code hash of the contract at the specified account id. It is not possible to use this function in the integration test environment.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/code-hash

Implementation Idea	Note: The following estimate applies both to code_hash() and to own_code_hash(), as the two functions are very similar.
	
	This function requires accounts and contract instantiation to be implemented. After that's done, there are two main methods to implement this functionality:
	
	1. Hash some property of the specified contract. For example, the address of the constructor.
	2. Implement the code hash equivalently to how it's done on E2E.
	
	Option #1 is fairly easy to implement. However, it can only support comparing hashes returned by either own_code_hash() or code_hash(), not if the caller specifies a hash directly. E.g.
	
	if code_hash(foo) == "0xf00..."{
	}
	
	Option #2 supports both use cases, but it's more complex to implement and also slows down integration tests by requiring the full WASM compilation of the contract.

Complexity Level	Complex, yet feasible
Implementation Time Estimate	5 days [FTE=1] (for Option 1, suggested) 15 days[FTE=1] (for Option 2)

Proposed Implementation Order	#5
Function	own_code_hash

Issue Number	10
Status	Missing Function Implementation on Integration Testing
Issue Description	`own_code_hash` retrieves the code hash of the currently executing contract. It is not possible to use this function in the integration test environment.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/own-code-hash

Implementation Idea	Note: The following estimate applies both to code_hash() and to own_code_hash(), as the two functions are very similar.
	
	This function requires accounts and contract instantiation to be implemented. After that's done, there are two main methods to implement this functionality:
	
	3. Hash some property of the specified contract. For example, the address of the constructor.
	4. Implement the code hash equivalently to how it's done on E2E.
	
	Option #1 is fairly easy to implement. However, it can only support comparing hashes returned by either own_code_hash() or code_hash(), not if the caller specifies a hash directly. E.g.
	
	if code_hash(foo) == "0xf00..."{
	}
	
	Option #2 supports both use cases, but it's more complex to implement and also slows down integration tests by requiring the full WASM compilation of the contract.

Complexity Level	Feasible
Implementation Time Estimate	1 days [FTE=1] (With instantiate_contract and code_hash implemented, this implementation should be straight forward)

Proposed Implementation Order	#6
Function	`invoke_contract`
Issue Number	4

Status	Missing Function Implementation on Integration Testing
Issue Description	It is not possible to make a call to another contract in integration test environment.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/invoke-contract
Implementation Idea	The function depends on features also required by gas_left(), instantiate_contract(); namely, storage and retrieval of secondary contracts, and gas calculations. The logic is fairly complex, so it's difficult to provide a time estimation. As a very rough estimate, it could be around 1-2 weeks worth of work. On-chain implementation at frame/contracts/src/exec.rs:1199-1243 On-chain implementation at frame/contracts/src/exec.rs:885-1014
Complexity Level	Complex, yet feasible
Implementation Time Estimate	10 days [FTE=1]

Proposed Implementation Order	#7
Function	`invoke_contract_delegate`
Issue Number	3
Status	Missing Function Implementation on Integration Testing
Issue Description	There is no possibility to currently use delegate calls in integration tests.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/invoke-contract-delegate
Implementation Idea	This function has a lot of overlap with invoke_contract. Once one of the two is implemented the other one should cost only a few more days.
Complexity Level	Feasible
Implementation Time Estimate	3 days [FTE=1]

Proposed Implementation Order	#8
Function	`caller_is_origin`
Issue Number	8
Status	Missing Function Implementation on Integration Testing
Issue Description	Integration testing is missing critical functionality to properly write tests using caller_is_origin.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/caller-is-origin
Implementation Idea	The function depends on features also required by gas_left and invoke_contract; namely, storage and retrieval of secondary contracts, and gas calculations. Assuming that those features are already in place, the function is fairly trivial. On-chain implementation at frame/contracts/src/exec.rs:1385.
Complexity Level	Feasible
Implementation Time Estimate	1 day [FTE=1]

Proposed Implementation Order	#9
Function	`set_code_hash`
Issue Number	6
Status	Missing Function Implementation on Integration Testing
Issue Description	Not implemented for integration test. Error associated to ink! version in E2E test. Would permit contract upgradability.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/set-code-hash

Implementation Idea	The function appears to depend on features also required by gas_left() and instantiate_contract(); namely, storage and retrieval of secondary contracts, and gas calculations. Until those functions are implemented, it's unfeasible to estimate an exact implementation cost, we offer an upper bound of 3 days. On-chain implementation at frame/contracts/src/exec.rs:1489
Complexity Level	Feasible
Implementation Time Estimate	3 days [FTE=1]

Proposed Implementation Order	-
Function	`gas_left`
Issue Number	5
Status	Missing Function Implementation on Integration Testing
Issue Description	Gas usage in the integration environment should have the same limitations as in the blockchain environment (end-to-end).
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/gas-left
Implementation Idea	Because integration tests are performed on native code rather than WASM code, and because gas cost is based on the number of WASM instructions executed, implementing this function is impractically complex.
Complexity Level	Impractical to implement
Implementation Time Estimate	-

Proposed Implementation Order	-
Function	`call_runtime`
Issue Number	11

Status	Missing Function Implementation on Integration Testing
Issue Description	Tries to trigger a runtime dispatchable, i.e. an extrinsic from a pallet. For more details consult host function documentation.
Detailed Documentation & Test Case	https://github.com/CoinFabrik/on-ink-integration-tests/tree/main/test-cases/call-runtime
Implementation Idea	Implementing this function is unfeasible, as it would require emulating practically the entire node to get consistent results. Testing a contract function that calls into the runtime should be done using E2E tests.
Complexity Level	Unfeasible
Implementation Time Estimate	-

Conclusions

From our list of 24 functions enumerated in the Attachment 1: Universe of Functions defined in our scope, we analyzed 11 in detail during our 2 week discovery and provided development time estimates for them.

Of these 11, 9 corresponded to functions which were identified due to their usage of the `unimplemented!()` macro for Integration tests, and 2 were functions which we had come across and found implementation differences during the development of fuzzers for Scout. We deemed as feasible the implementation of 9 out of the 11 analyzed functions.

The remaining 13 functions in the Attachment 1: Universe of Functions show an implementation both for Integration tests and E2E tests. In order to assess implementation differences in these functions, smart contract examples can be created evaluating each one of them. Functions whose testing shows differences can be analyzed further, and their implementation effort necessary to correct these differences can be estimated, as demonstrated in the case of default_account() or set_contract_storage().

Next Steps

In Milestone 1 we determined there are 9 functions to work on with explicit plans, and 13 functions, for which there might be implementation differences that remain to be analyzed. For our next milestone, we will focus:

- On the implementation of the unimplemented functionalities for integration tests found on Milestone 1, as well as the resolution of implementation differences between integration and e2e tests.
- We will also review the remaining 13 functions, which are implemented both for integration and e2e tests, and correct their implementation differences.
- QA: We will add tests to integrate the functions we added or modified to the ink! project repository.
- Report Describing our Contribution. This report will also include a basic categorization of smart contracts (e.g: DeFi, AMM, etc) and an according classification of our test cases.