Mini Pulp Process

Test Documentation

Version 1.2.0.0

Eero Eriksson
Document state: Complete Modified: 24.3.2025

CONTENTS

1 INTRODUCTION

This document is the test documentation concerning the Mini Pulp Process simulator. This is specifically about the software’s server side of the process known as BatchProcSimulator. The purpose of this document is to explain the process used to test the software as well as what parts of the software are tested and why. With this document and the test files made during the development, the user should have a broad understanding of the current testing process of the simulator.

2 TESTING ENVIRONMENT

In this chapter the scope of the testing and the criteria that the software must pass to be considered to be correctly functional, are defined. Additionally, the techniques and the tools that were used to test the software are defined so that the tests can be replicated or improved when necessary.

2.1 Requirements

Functions and classes are tested during Unit Testing (UT). Classes Equipment, Tank, PressureTank and Simulation are the classes that are tested as a whole as well as all their public methods. This also applies to the module Routes with additional tests on variables used in the OPC UA connection which must be correct to find the client software. The goal is having a 100 % coverage on all the branches and functions as an unexpected error will most likely crash the software.

Integration Testing tests that the interactions between the classes and modules work as intended (IT). The number of interactions is kept as low as possible to keep the software easier to maintain. Therefore, all these connections are critical to the system.

During the System Testing (ST), the software's functionality is tested as a complete product in connection with the client. The simulator must apply all the allowed controls given by the client to the server. The connection between the server and the client must be stable while the software is in use excluding the known issue of using the validation version of the OPC UA stack on the client's side, meaning the software can only be run 1 hour at a time.

The results of the system testing are validated during User Acceptance Testing (UAT) to see if the end user can operate the software without problems. The instructions on how to use the software must be easy enough for the user to understand. Most of the users of this software are not previously familiar with it so the ease of usage is a vital aspect.

2.2 Approach

Unit and integration tests are done by using Jest. Jest is a JavaScript testing framework. It works the same way as other NPM packages, it is added to the package.json file as a dependency. In this case, the version used to make the tests is 29.7.0. Package.json has a script to run all the available tests so it can be run by typing "npm test" in the terminal of the working directory.

Unit tests are made by excepting predetermined output with different input. The inputs include both correct and incorrect input to test the software's robustness. In some cases, a variable's type is also checked. After the functions are tested individually, there are a couple of randomized combinations to mix different methods that impact the class's main output.

The integration tests are done to observe the changes in the Equipment class after running the simulation as a whole. The Equipment class is the link between the servers and the logistics of the simulator. The class is fed with common user controls to see if the output sent to the user interface is as it should when running different amounts of simulation cycles. The JSON string contains the states of the actuators so it can be used to check both the information for the frontend as well as the user's control system. It also includes a test to see if the OPC UA connection works as intended.

During the system testing the simulator is used as a complete product to ensure it meets all the requirements. This is done by running different scenarios and comparing the results to the result given by the old version. Unfortunately, there is limited data available about the result the physical system gives which could be used in this phase of the testing.

After the software has passed all the other tests, the software is delivered to the end users for User Acceptance testing to ensure that the software is easily usable to the user and has no conflicts with the user systems, like operating systems.

3 TEST CASES

All the test cases and their dependencies are introduced in this chapter. The different cases are divided into groups based on the level of testing.

3.1 Naming of test groups and cases

The name of the test group consists of the group and the index number of the test within the group. The groups are Unit Test (UT), Integration Test (IT), System Test (ST) and User Acceptance Test (UAT). The identifiers in this document should be found in their respective testing codes. For example, the first case of the Unit Test group is named UT:01.

3.2 Relationships between test cases

The test cases are related to one another. The relationships between the cases are displayed in figure 1 and 2 below. They are separated between the unit tests and the rest of the tests that are of higher-level. The assumption is that the unit tests are completed before the other tests.

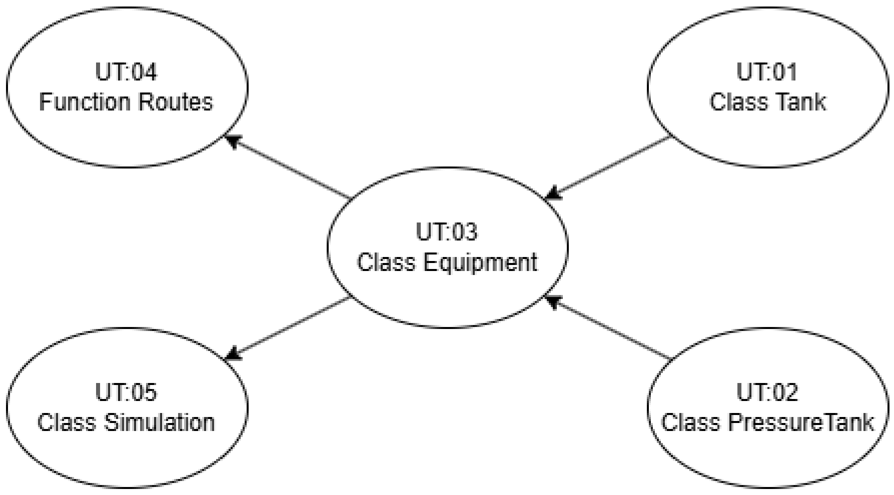

Figure 1. Relationships of the unit tests

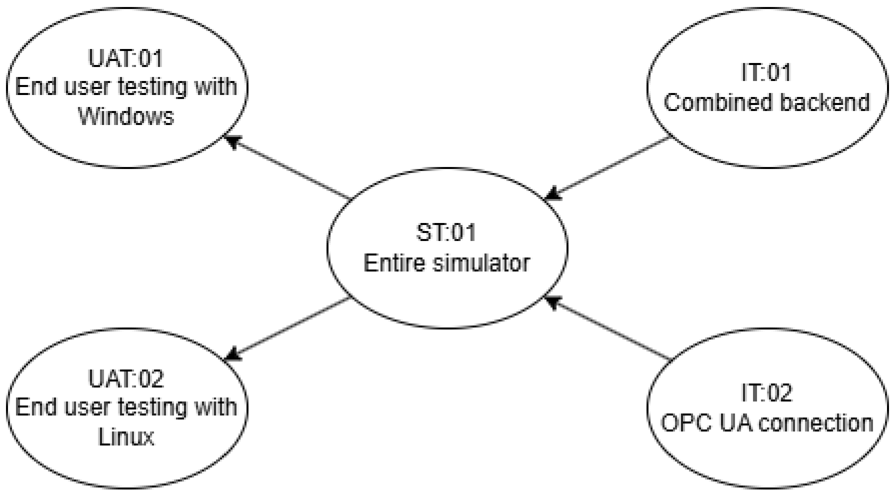

Figure 2. Relationships of the higher-level tests

3.3 Unit test cases

In this subchapter, there is a more precise list of unit tests made to the simulator and the parameters related to them.

3.3.1 Test group UT: Case 01

Identifier	UT:01
Description	Tests the methods of class Tank
Prerequisites	None
Inputs	The flow in and out of the tank, the temperature of the flow in and the state of the heater.
Outputs	None but the values stored in the class are used elsewhere.
Severity	High, will not break the simulator but will highly reduce its functionalities.
Probability	Very low.
Additional information	Test files: Tank.test.js

3.3.2 Test group UT: Case 02

Identifier	UT:02
Description	Tests the methods of class PressureTank.
Prerequisites	None

Inputs	Check for active pumps, the flow in and out of the tank, the temperature of the flow in, check for escaping pressure, state of the throttle valve and the amount of throttling.
Outputs	None but the values stored in the class are used elsewhere.
Severity	High, will not break the simulator but will highly reduce its functionalities.
Probability	Very low.
Additional information	Test file: PressureTank.test.js

3.3.3 Test group UT: Case 03

Identifier	UT:03
Description	Tests the methods of class Equipment.
Prerequisites	An instance of Equipment must be made and the method initializeActuators() must be the first method to be called for the data structure to be initialized. Uses classes Tank and PressureTank.
Inputs	None, name of an actuator and value of an actuator.
Outputs	Incorrect inputs are expected to throw the correct error, correct inputs are expected to return the right output (both value and type).
Severity	Severe, the simulator crashes
Probability	Should not be possible.
Additional information	Test files: Equipment.test.js

3.3.4 Test group UT: Case 04

Identifier	UT:04
Description	Tests the methods of function Routes.
Prerequisites	An instance of Equipment must be made and the method initializeActuators() must be the

	first method to be called for the data structure to be initialized.
Inputs	Data structure containing the actuators.
Outputs	List of possible routes for the fluid to move in the system.
Severity	Severe, the simulator relies on the routes to be correct to function as intended.
Probability	Very low.
Additional information	Test file: Routes.test.js

3.3.5 Test group UT: Case 05

Identifier	UT:05
Description	Tests the methods of class Simulation.
Prerequisites	An instance of Equipment must be made and the method initializeActuators() must be the first method to be called for the data structure to be initialized.
Inputs	Data structure containing the actuators.
Outputs	Updated data structure containing the actuators.
Severity	Severe, if this class doesn't work, the simulation cycle does nothing. The software keeps going but does nothing useful.
Probability	Very low.
Additional information	Test file: Simulation.test.js

3.4 Integration test cases

In this subchapter, there is a more precise list of integration tests made to the simulator and the parameters related to them.

3.4.1 Test group IT: Case 01

Identifier	IT:01

Description	Tests the performance of the parts of the backend as a united solution.
Prerequisites	Classes Equipment, Simulation, PressureTank, Tank and function Routes.
Inputs	States of the actuators that would have been assigned by the control system.
Outputs	The states of the actuators as a JSON string that is used in the frontend. Also, the information is available to the control system but in different formats.
Severity	Severe, this is the core function of the backend so malfunction will make the system drastically less useful.
Probability	Very low. Highly related to the results of the unit tests, if they work, this will most likely work as well.
Additional information	If a case of failed test, check the result for rounding errors as it is a JSON string. Test file: Integration.test.js

3.4.2 Test group IT: Case 02

Identifier	IT:02
Description	The OPC UA connection between the server and the client.
Prerequisites	Functional server (UaServer) and Equipment class and client (tuni.MppOpcUaClientLib).
Inputs	Different actuator states.
Outputs	Information is transferred successfully between the two.
Severity	Severe, this is a requirement to use the simulator.
Probability	Low.

3.5 System test cases

In this subchapter, there is a more precise list of system tests made to the simulator and the parameters related to them.

3.5.1 Test group ST: Case 01

Identifier	ST:01
Description	Tests the whole simulator's
Prerequisites	All previous tests are successful, and all the parts of the software are available. Needs an OPC UA client to change the states of actuators.
Inputs	The control by the client.
Outputs	Same result as the previous version of the simulator gives.
Severity	Severe, If the result doesn't match, the software doesn't meet its requirement.
Probability	Very low, if the previous tests are successful, so should this one be.
Additional information	Check for rounding discrepancies.

3.6 User acceptance test cases

In this subchapter, there is a more precise list of user acceptance tests made to the simulator and the parameters related to them.

3.6.1 Test group UAT: Case 01

Identifier	UAT:01
Description	End user testing with Windows OS
Prerequisites	A person with Windows OS who is familiar with the simulator as well as a person that isn't familiar.
Inputs	The software with all the necessary documents that are given to the end users.
Outputs	The person can use the software independently.

Severity	Severe, this is the main use case of the software.
Probability	Low

3.6.2 Test group UAT: Case 02

Identifier	UAT:02
Description	End user testing with Linux OS
Prerequisites	A person with Linux OS who is familiar with the simulator as well as a person that isn’t familiar.
Inputs	The software with all the necessary documents that are given to the end users.
Outputs	The person can use the software independently.
Severity	Severe, this is the main use case of the software.
Probability	Low