Improvement of Test Automation Process

Devecto Oy

Anzhelika Kettunen

Bachelor's thesis
April 2019
Technology, communication and transport
Degree Programme in Information and Communications Technology

jamk.fi

Description

Author(s) Kettunen, Anzhelika	Type of publication Bachelor’s thesis	Date April 2019
		Language of publication: English
	Number of pages 58	Permission for web publi-cation: yes
Title of publication **Improvement of Test Automation Process**		
Degree programme Information and Communications Technology		
Supervisor(s) Ari Rantala		
Assigned by Devecto Oy		

Abstract

The main goal was to develop a new method to automate quality assurance processes and available tools for its realization, which would significantly increase labour productivity. The project aimed to create a platform that enables to automate the testing routines and reduce the amount of manual testing.

The structure of CI system, its components and variants of available tools for implementation were researched. A new system able to automatically create test routines from a test drive test run is described here. Justification of the selected technologies and the description of the implementation were carried out as well.

The principles of the script creation software, concept and logic were developed 100% within this project. As a result, the application was designed and programmed on C#. The demo version demonstrated the testing capabilities and basic requirements of the example of implementation and usage script creation software in HIL testing of heavy moving machinery. The purpose of the application was to define the necessary data and parameters to enable creating scripts based on signal changes.

The developed application allows seeing signals of real machines during manual test execution, saving signals’ values and generating robot scripts based on them. The application was also successfully tested in real environment.

Keywords/tags (subjects)
Robot Framework, HIL, ALM, Continuous Integration, CI, Testing

Miscellaneous (Confidential information) jamk.fi

Kuvailulehti

Tekijä(t) Kettunen, Anzhelika	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Huhtikuu 2019
		Julkaisun kieli Englanti
	Sivumäärä 58	Verkkojulkaisulupa myönnetty: kyllä
Työn nimi **Improvement of Test Automation Process**		
Tutkinto-ohjelma Tieto- ja viestintätekniikka		
Työn ohjaaja(t) Ari Rantala		
Toimeksiantaja(t) Devecto Oy		

Tiivistelmä

Tavoitteena oli luoda alusta, jonka avulla toimeksiantajan asiakkaat pystyvät automatisoimaan ohjelmiston testaamiseen tarvittavien testirutiinien tekemisen ja vähentämään kuormittavaksi havaittua manuaalitestaustyötä. Tämä oli tarkoitus toteuttaa ohjelmistorobotiikan keinoin siten, että manuaalista testausta tekevät henkilöt voisivat suorittaa kertaalleen manuaalisen testin ja tallentaa saman rutiinin tämän jälkeen testiautomaation suoritettavaksi. Päätavoitteena oli luoda ohjelmisto testausrutiinien automatisointiin sekä manuaalisen testauksen määrän vähentämiseen.

Työssä tutkittiin CI-järjestelmän rakennetta sekä sen komponentteja ja työkaluja. Lisäksi kuvailtiin uutta järjestelmää, joka pystyy automaattisesti luomaan testirutiinit koeajotestistä, hakemaan luodut robot-skriptit versiohallinnan palvelimelta, suorittamaan robot-testit HIL-simulaattorilla ja raportoimaan testituloksista. Tehtiin valittujen teknologioiden perustelut ja toteutuksen kuvaus.

Ohjelmiston periaatteet, konsepti ja logiikka kehitettiin täysin projektin sisällä. Sovellus toteutettiin C#-ohjelmointikielellä käyttäen Microsoft Visual Studio 2017 ja TAPI dll-kirjastoa, joka oli valmistettu testauskoneen mallin ominaisuuksien perustella. Sovellus saa TAPI-kirjaston avulla testauskoneen signaalitiedot, tallentaa ne, analysoi ja generoi robot-skriptit annetun sapluunan ja ehtojen perustella. Demo-versio kuvasi testausominaisuuksia ja perusvaatimuksia. Esimerkkinä oli raskaiden liikkuvien koneiden HIL-testaus.

Sovellus testattiin onnistuneesti todellisessa ympäristössä.

Avainsanat (asiasanat)
Robot Framework, HIL, ALM, Continuous Integration, CI, Testing

Muut tiedot

Contents

Figures

Tables

Acronyms

ALM	Application Lifecycle Management
API	Application Programming Interface
CAN	Controller Area Network
CI	Continuous integration
CVS	Concurrent Versions System
DOM	Distributed Object Model
DUT	Device/Unit under test
ECU	Electronic Control Unit
EDT	Electronic Diagnostic Tool
HIL	Hardware-in-the-loop
HTTP	Hypertext Transfer Protocol
JRE	Java Runtime Environment
RMI	Remote Method Invocation
SCM	Software Configuration Management
SUT	System Under Test
TAPI	Test Application Programming Interface
XML	eXtensible Markup Language

1 Introduction

1.1 Background

The world is moving forward and the current technological progress affects people in very different areas. It changes the way they live and the way they work. Digitalization allows people to use information more efficiently, which leads to the fact that one can easily find new ways to develop it. Information processing can be applied to almost all areas of the business world. New innovative solutions have a big power on the market nowadays. More and more complex machines are developed, and the competition is growing. One of the priorities of production companies is to develop safe, high-quality products. All the time companies try to speed up the development, automate testing processes. This study discusses one of the ways to improve the automated testing process.

This project is assigned by Devecto Oy. Devecto is a software R&D company specialising in demanding software development for smart machines and devices.

Devecto's business consists almost entirely of product development related to software design for industrial product development. The design process varies slightly from one customer to another; nevertheless, the software design project that is typical in its content usually covers software requirements, implementation, and quality assurance. (Devecto Oy 2019.)

For easier understanding Figure 1 represents Devecto’s generalized groups of services on the diagram.

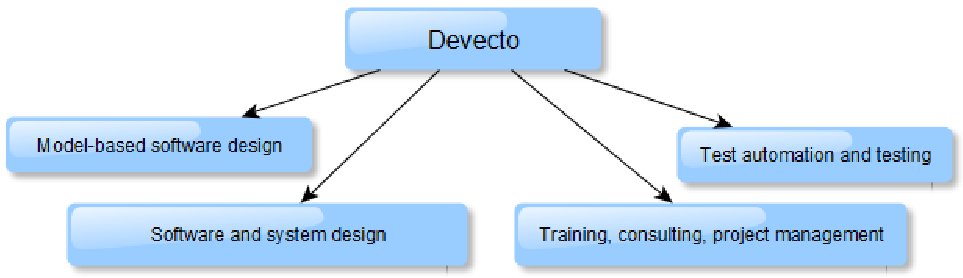

Devecto
Model-based software design
Software and system design
Training, consulting, project management
Test automation and testing

Figure 1. Devecto's Software Development Services

As can be seen in Figure 1, the services affect different areas, such as design, testing and consulting. Devecto takes care of the quality job in software architectures, embedded software design, model-based software design, software testing and software testing development, functional safety, user interface design and IoT solutions. Hence, working as an employee at Devecto gives a good opportunity for self-development from student to specialist.

1.2 Goals of study

The main goal of this study is to develop a new method to automate quality assurance processes and available tools for its realization, which would significantly increase labour productivity.

Quality assurance is currently being made automatized so that new software changes can be tested in as agile method as possible. Automation also enables the so-called regression testing for the entire software, so that the integrity and dependencies of the big entity can be verified.

The project aims to create a platform that enables to automate the testing routines and reduce the amount of manual testing. This is supposed to be implemented by software robotics so that operators doing manual testing could perform a manual test only once and save data of a performed routine for future run of the same test, yet already on a simulation platform.

For example, in situations where new software changes or repairs occur in the software at the finishing stage of development, the test periods can last for several days, including sections that could be easily exported to automation. If test runs were automated, it would safe working hours for the most demanding tests, and greatly speed up the test cycle. In this case, customers get their products faster on the market and project payback times are reduced.

Writing test routines manually is almost as laborious as doing the actual software. Performing test routines for example on rolling stock projects on the real machine is much faster than writing a routine.

This paper discusses and describes a new system that should be able to automatically create test routines from a test drive test run and structure of CI system, its components and variants of available tools for implementation.

2 Available implementations (research)

This study examines different kinds of technologies are examined, which can be used for the required system implementation. In the next chapters, the principle of operation simulation system, test automation frameworks, continuous integration servers and Application Lifecycle Management Software are discussed. Figure 2 presents how the system should be organized in details. Based on this information, the development of a new automatic test creation software is implemented. This figure also demonstrates the connection of automatic test creation software with continuous integration block.

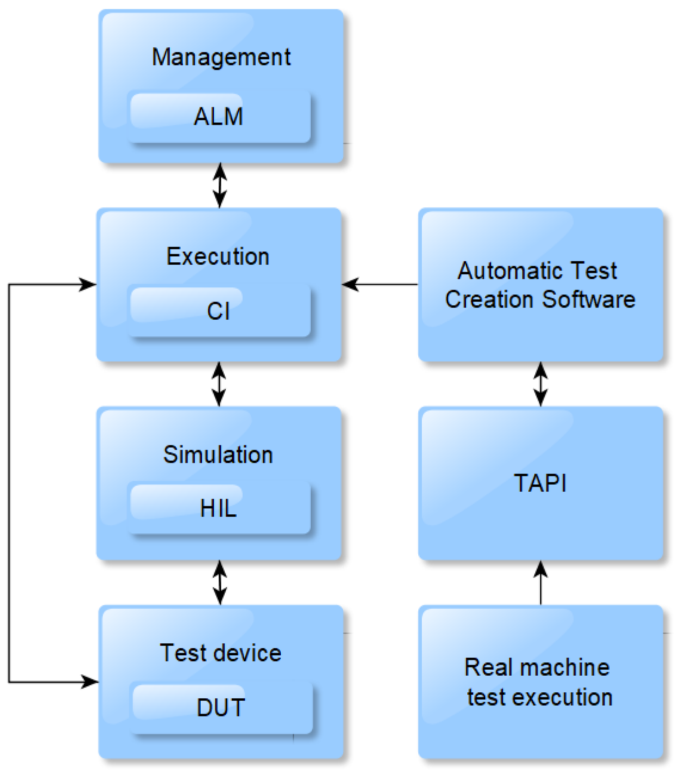


Figure 2. Goal architecture

Development of a project can be managed via Application Lifecycle Management software, which can contain the specification of product, stages of development, testing and maintenance information.

Continuous integration server is connected with ALM, as the design of test execution takes place based on the requirements; the project management also implies management of test results; hence, they should be provided from CI to ALM.

Usually the mediator between CI server and Simulator is test automation framework, which provides the execution environment for automation test scripts.

Simulation system (hardware parts and software) represents the behaviour of a simulated machine, its response on conducted in tests actions. The behaviour of the simulation system must meet the requirements (the requirements are manage in ALM).

The goal of this project is to get a system/software that could generate ready test scripts based on manual tests, which could be used in a common automated test process. Generally, the system should get the opportunity to save test scripts for future automatic execution.

Automatic Test Creation Software should be able to communicate with TAPI. During the manual machine testing, TAPI can send operator events to test creation software, which processes the events and generates the test scripts for the simulation model. An additional advantage of this system is that the run results can be also used for verification of simulation quality (see more information in Chapter 6 “Simulation system”).

3 Application Lifecycle Management Software

3.1 About ALM phases

Application Lifecycle Management covers the entire lifecycle of a product from conception through retirement. ALM includes the specification, design, development, and testing of a software application, and also deployment and maintenance. Figure 3 represents the sequence of Application Lifecycle Management phases.

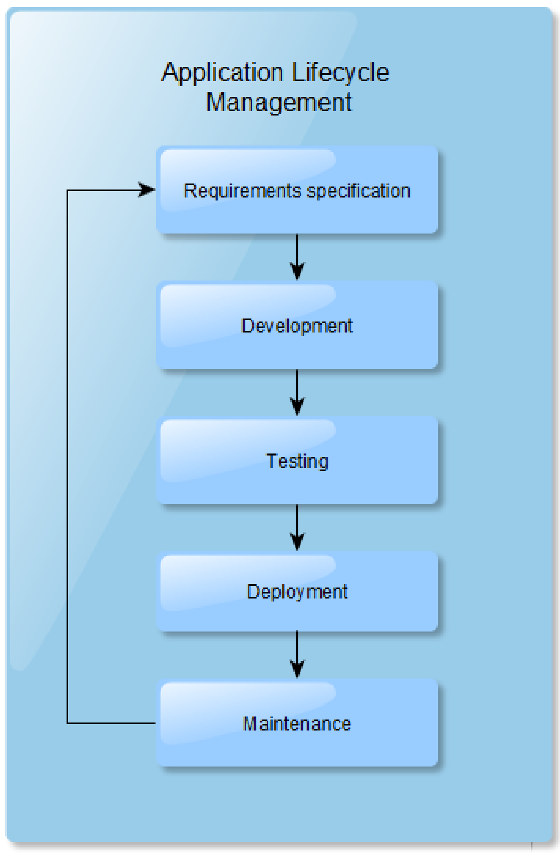


Figure 3. Application Lifecycle Management phases

Any application’s lifecycle starts from the definition of requirements. The first phase, Requirements specification, can also include development of priorities. Requirements can have a hierarchical tree structure, where each node is a sub-requirement for its larger parent node. Prioritization can be performed as classification of requirements, e.g. what is a must or what can come later.

As for testing, preparing of environments and test-cases for testing phase should start during the development phase. The testing team already at this stage can provide feedback on the software, contact the developers and help them to validate the features if needed.

Once the development is completed, the full integration testing of the entire system should be done. The main task of the testing phase is to verify that the application complies with the defined requirements. At this stage, the testing team runs their verifications of all the features developed. The testers identify and report issues that need to be addressed by development.

The quality of software mostly depends on the work done in the previous phases. The better planned the requirements, the closer the system is to the actual needs of the user. The more issues were caught in development and testing phases, the fewer problems there are during maintenance process. (Application Lifecycle Management Software 2019.)

3.2 JIRA

JIRA Scrum Framework helps developers in collaboration and management of complex projects. Jira Scrum Board provides a visual display of progress during the development cycle. Jira Scrum Board could be a very good choice for highly technical teams who practice agile project management, in especially if the most important aspects are: (Средства для разработки ПО. Jira Software 2019)

- Team members’ communication and transparency of progress.
- Organizing the work around the sprint timeframe.
- Transparency in the team's work by utilizing burndown and velocity reports.

In the author’s opinion, the strongest feature of Jira is an easy way of tasks and bugs tracking. However, this project is designed for realization of test automation for big and complicated systems, which have a large number of requirements and development stages. At least the author did not find among Jira’s opportunities an easy enough way of managing the simulation of complicated.

3.3 Polarion

Polarion ALM, Application Lifecycle Management platform provides requirement management, reuse and branch management, configuration management, and reporting. Polarion helps to handle and support requirements, design, testing and tracking defects, improve application development processes as well as coding, testing and release. It also allows to gather, make approvals, manage requirements, and link them to user stories even for complex systems during all project lifecycles. Polarion has browser-based access and provides easy way for editing of specification documents. (Polarion ALM 2019.)

Polarion ALM is a good tool for big complicated *projects*, especially if several teams with quite different tasks are working on the same *project.*

3.4 Chapter summary

In previous chapters, some ALM systems were described, which helps to understand the differences between solutions and main features. Now it can be concluded, that ALM for test automation system should be chosen based on the possibility to provide the next features and possibilities:

- Should provide functionality to help estimate and plan projects.
- It could be useful, if use cases were stored in the system and had links to the requirements.
- Create, edit and manage test cases.
- Track bugs, enhancements and issues linked to test cases.
- Record bugs linked to test steps for full test traceability.
- Manage both manual and automated testing.
- Ability to attach documents, screenshots and URLs to all artefacts.

On the base of information above, it can be concluded that Polarion is a more suitable solution for big complicated projects. The negative sides of Polarion in comparison with Jira is that Polarion is harder to learn to use, when Jira (this decision was made on the base of the writer's own experience); however, it is not a big problem especially if it is taken into use for big, long lasting projects.

4 Continuous Integration Servers

4.1 About Continuous Integration process

Continuous integration (CI) is the development practice of merging development code to a shared repository as often as developers need it done. It is an important part of the modern software development life cycle.

During software development process a new code can be added, the existing code can be modified or deleted. Therefore, it is critical, that changes will not have a negative impact on already existing code elements. With this reason often automated testing is an inherent part of the continuous delivery process. Ideally the whole process of software development and all associated actions should be automated, which makes the process much more efficient. In a case if test results show good results, the tested software package can be, e.g. deployed in production. Otherwise development team members should take into account errors detected during testing. (Continuous integration 2019.)

Figure 4 represents the cycle of Continuous Integration processes on diagram. It starts from the planning and ends on the monitoring process, after that the next cycle starts from the planning and repeats the same way again and again.

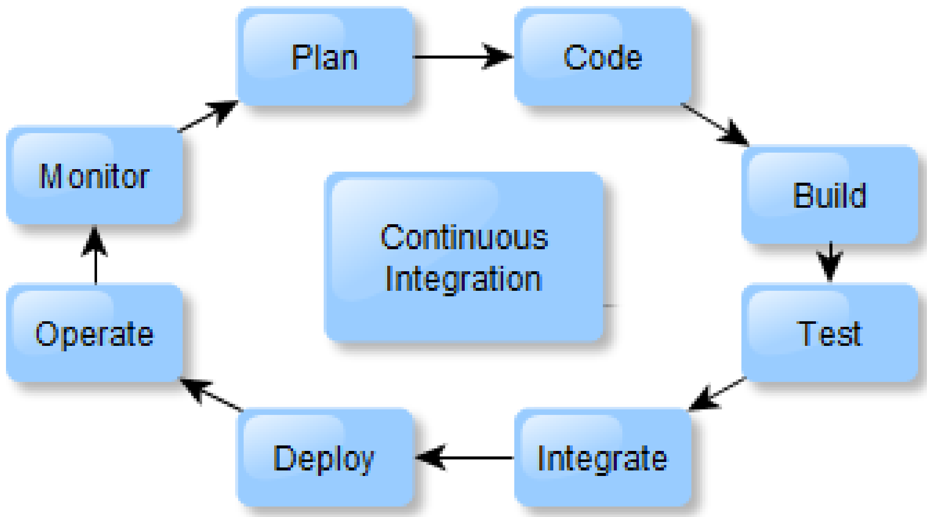


Figure 4. Continuous Integration process

Every year software development and testing techniques are becoming more and more complex. To orchestrate the whole developing process, to ensure that the steps required for software delivery are being executed in the correct order and that failures are handled in a correct way, for all these purposes Continuous Integration Server is used. Next, some of the most popular solutions are considered and reviewed from the perspective of integrating performance testing into the continuous integration and delivery process.

4.2 Jenkins

Jenkins is an automation server platform that provides automated builds, unit tests, SW deployments, HIL tests and reporting. The benefit of Jenkins is that it is Java-based continuous integration tool and it is free, open source and modular. One of the benefits is also the easy installation through native system packages, Docker or with JRE. Figure 5 represents Jenkins in action. Web-Base user interface, opportunity to run several jobs in the same time. The dependence between triggers and jobs can be set. Jenkins jobs communicate over TCP/IP with job executors. Postexecution also can be configured based on execution results.

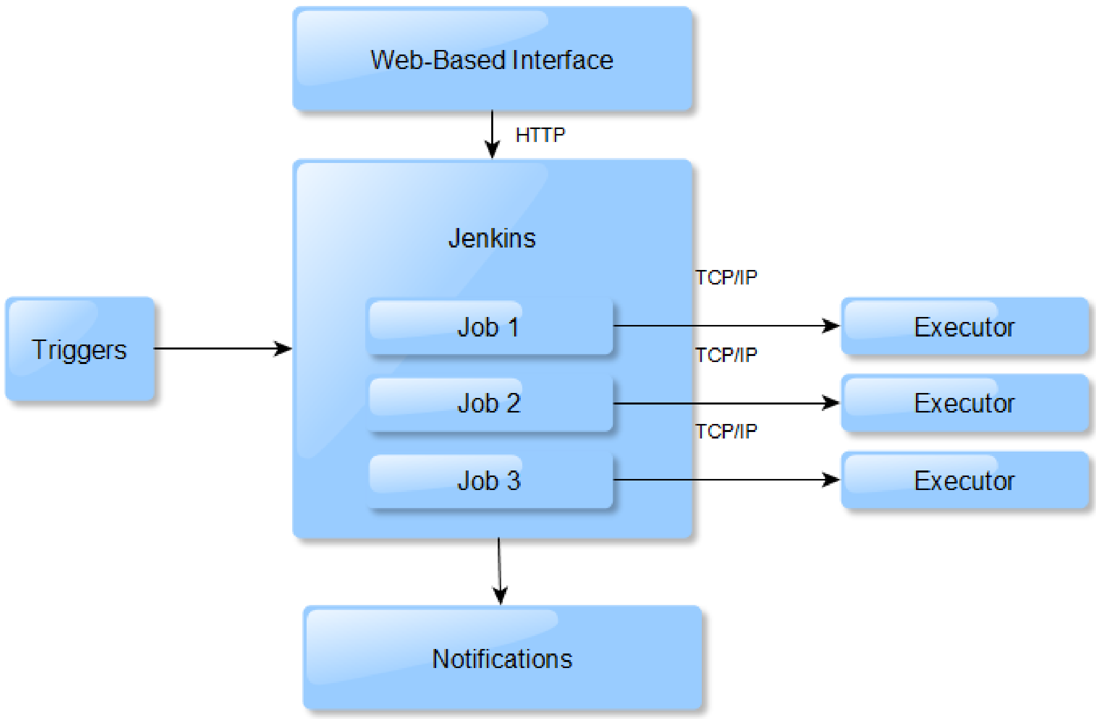


Figure 5. Jenkins in action

Jenkins' functionality can be extended through the installation of plugins. It supports SCM tools, such as CVS, Mercurial, Git, Subversion; it can also execute arbitrary shell scripts and Windows batch commands. There are more than 1,000 plugins providing extra integrations to support building and automating any project. (Jenkins User Documentation 2019.)

4.3 Buildbot

Buildbot is a continuous development and build server. It supports many integrations, distributed builds, custom build steps, notifications and others features. Buildbot is some sort of a scheduling system; it can queue jobs, execute the jobs depending the required resources are available, and report results. Buildbot installation has one or more masters and several workers. The master can coordinate the activities of workers, monitor source-code for changes, and report the results to developers. The operation principle is presented in Figure 6.

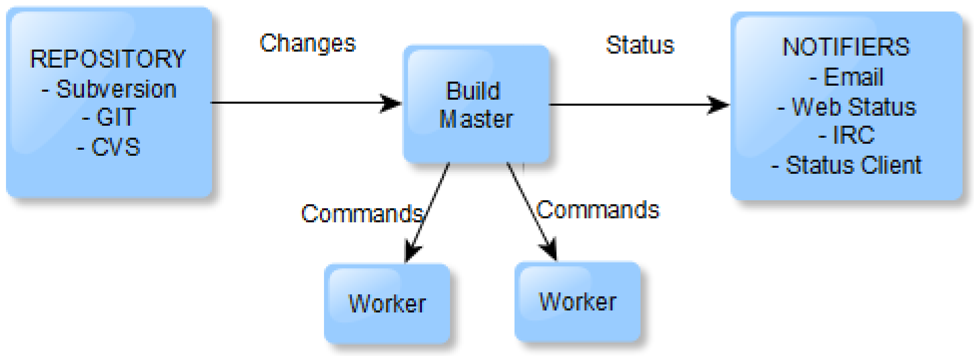


Figure 6. Buildbot operating principle

Buildbot is a Python-based tool, and the config files are provided to the master as very simple Python scripts, which allows dynamic generation of configuration and customized components. (Buildbot Basics. N.d.)

4.4 CruiseControl

CruiseControl is one of the oldest CI tools; at the same time it is an extensible framework for creating a custom continuous build process. Web-based interface provides details of the current and previous builds. The configuration is XML-based. CruiseControl is available in three distributions: binary distribution, source distribution and windows installer. Figure 7 represents the architecture diagram, where can be seen that HTTP and RMI technologies allow to monitor and control the CruiseControl build loop.

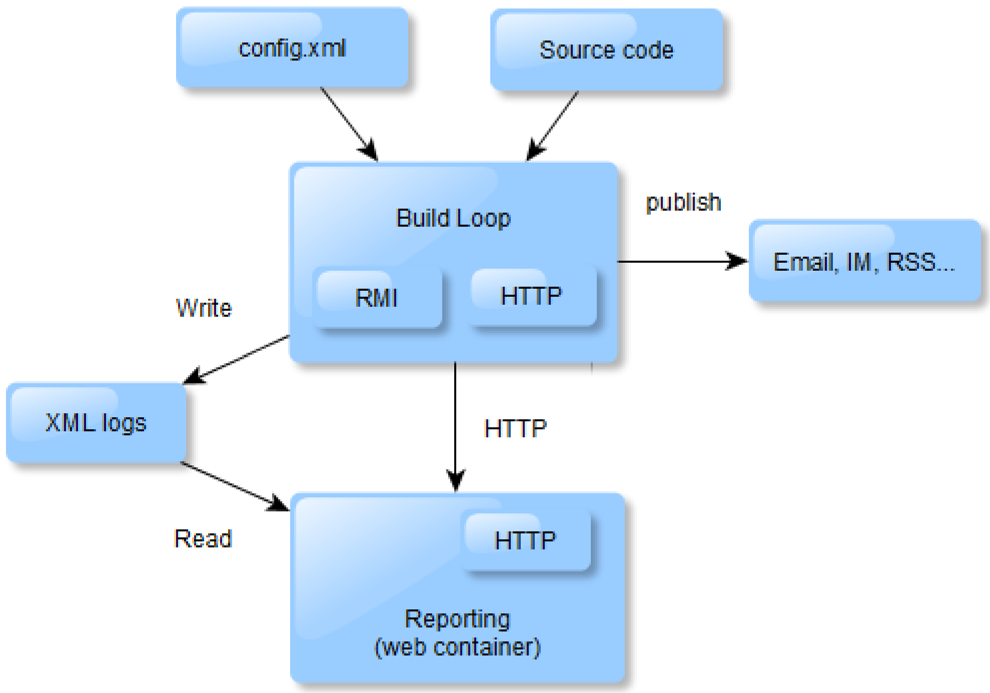


Figure 7. CruiseControl architecture diagram

The Build Loop runs as a scheduled daemon process, checks changes in codebase, builds and sends notifications regarding the build. (CruiseControl 2019.)

4.5 Chapter summary

Polarion Jenkins Connector is Polarion Extension, which provides connection between Polarion and allow to trigger a Jenkins job, to notify Polarion about job results, and automated Jenkins test results can be imported as Polarion Test Runs. (PJ – Polarion Jenkins Connector 2016.) Therefore, Jenkins is a good choice as a continuous integration tool.

5 Test Automation frameworks

5.1 Types of Test Automation Frameworks

Testing frameworks are quite an important part of successful automated testing process. The main goal of test automation frameworks is to provide an execution environment for automation test scripts. It helps to reduce testing efforts, expenses, minimize manual intervention, increase test speed and efficiency. There are different types of automated testing frameworks, so developer should take into account their architecture, benefits and disadvantages, to develop, execute and reports efficiently. (Автоматизированное тестирование 2018.)

Figure 8 demonstrates 6 types of Test Automation Frameworks on diagram.

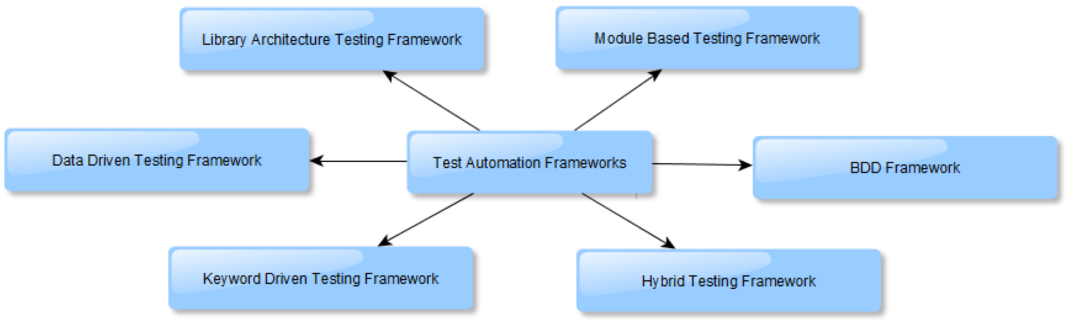

Library Architecture Testing Framework
Module Based Testing Framework
Data Driven Testing Framework
Test Automation Frameworks
BDD Framework
Keyword Driven Testing Framework
Hybrid Testing Framework

Figure 8. Types of Test Automation Frameworks

In brief, the list below covers Test Automation Framework Types: (Test Automation Frameworks 2019)

Modular Based Testing Framework: the application under test should be divided into separate units, which will be tested separately.

Library Architecture Testing Framework is based on modular framework but uses libraries of common functions that can be used by multiple test scripts.

Data-Driven Framework: The test scripts are connected to the external data source and can read and populate the necessary data when needed.

Keyword-Driven Framework: The functions of the tested application are saved in a table with a series of instructions in consecutive order for each test that needs to be run.

Behaviour Driven Development Framework, also known as Linear Automation Framework: There is no need to write custom code; the tester just records each step and then plays the script back automatically to conduct the test. This is one of the fastest ways to generate test scripts. Recording takes place quite fast; however, the data is hardcoded into the test script and tests cannot be re-run with multiple sets. If the data is altered, tests scripts also have to be modified.

Hybrid Test Automation Framework is a combination of any of the previously mentioned frameworks.

This information can be useful on the stage of choosing testing software. The next chapters describe some of the most popular test automation frameworks.

5.2 Robot Framework

The Robot Framework is an open source Python-base software and quite useful especially if Python is used for test writing; anyway, also IronPython (.NET) can be used or Jython (Java). Robot Framework is an application and it is operating system independent. This product uses a keyword driven approach that makes tests easy to read, understand and create. It includes many tools, most of which are developed as separate projects; however, there are also tools built into the framework. The libraries can communicate with the system directly. Robot Framework is easy to use; a test execution is started from the command line and the test report will be provided in HTML or XML format. (Robot Framework 2019)

5.3 Cypress

Cypress is an open source test runner built for the modelling web. Using Cypress it is easy to run end-to-end functional tests for anything that runs in a browser. Cypress is developer-friendly and focused on making test-driver development easier in order to understand everything happening inside and outside the browser. Cypress allows to add a debugger in a developing application. The developer has native access to every DOM element. The installation is quite simple, there is no configuration needed, no dependencies or changes to the code are required. Cypress allows to write tests quickly and watch them executed in real time. (Cypress 2019)

5.4 Citrus Framework

This open-source framework can help to automate integration tests for data format or any messaging protocol. Figure 9 demonstrates a diagram of typical test scenario. Citrus Framework is the most useful tool to test messaging integration in cases, when HTTP, REST, SOAT, FTP, SSH, JSON, XML or JMS messaging transport are used.

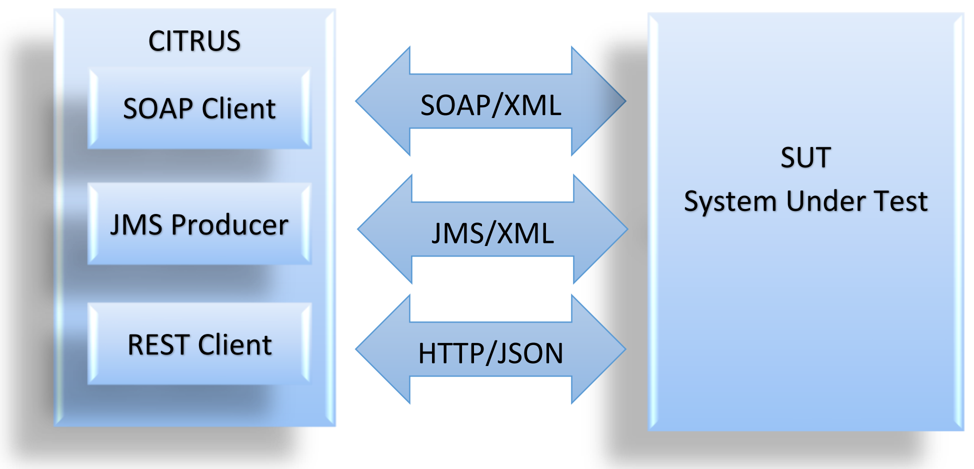


Figure 9. Typical test scenario (Citrus)

The system under test (deployed on an application server) can interact with Citrus over various message transports. The test is fully automated; hence, the integration tests can be added to continuous build. (Citrus Integration Testing 2019)

5.5 Chapter summary

Probably Robot Framework is the best solution for this project. It is especially quite useful that developers can create their own higher-lever keywords using the same syntax as for creating test cases; hence, the modular architecture can be extended, and also open source test libraries can be used.

Cypress can be useful in this project in case of developing a web-based user interface for test automation software.

Citrus Framework is suitable for testing communication between software and TAPI, however, due to the limited time (script creation software should be developed within one month), no communication protocols were developed inside of this project.

At this stage, the goal can be defined more detailed: test automation software should create Robot scripts, which can be used in simulator testing.

6 Simulation system

6.1 Purposes

The main goal of simulation system creation is to get a system, which could model a target vehicle, plane or other technical device and be used in software testing. Ideally, this system has to return the same results or at least the results have to be as close as possible to the results of a real device. For example, a tractor simulation system represents some sort of imitation of tractor as a machine. The creation of a simulator requires a well-defined description of the machine, its characteristics and behaviour in different situations. Simulations can be used in developing processes, for testing and performance, to show the machine reaction and behaviour in alternative conditions and actions. Usually, simulating system consists of software model and hardware parts. (RD-hankkeet ja simulointi N.d.)

6.2 HIL (Hardware-In-The-Loop)

Hardware-In-The-Loop is a useful technique during development and test processes. HIL includes electrically emulated sensors and actuators. The value of each sensor is controlled by the software environment and can be read by the system under test. HIL testing offers an excellent alternative to traditional testing methods.

HIL simulation is mostly used in developing and testing of complex control, protection or monitoring systems. In case of traditional testing on a physical equipment, it can be inefficient, expensive, take much time and finally it can be unsafe. Figure 10 demonstrates connections between workstation, HIL simulator and ECU.

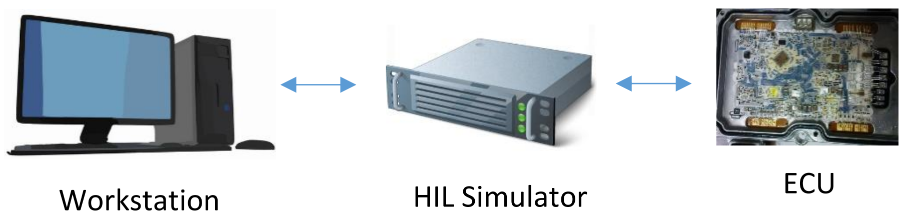


Figure 10. Hardware in the loop simulation

The HIL hardware is connected to the control electronics (DUT), where the software component to be tested is installed.

This technique differs from real-time simulation by the addition of a real component into the loop, which makes the development process more effective. Some real hardware components can be used instead of simulated ones due to the ease of use and lower expenses.

Today, Hardware-In-The-Loop is quite important in aerospace, power systems, automotive, power electronics industries, also it helps to meet the security requirements of critical infrastructure systems, which can be vulnerable to cyberattacks. The use of HIL improves the quality of testing, increasing the volume of the tested processes. (Программно-аппаратное моделирование 2018.)

6.3 ECU

In automotive electronics, the term Electronic Control Unit is used for designation of a device that can control the electrical systems in a machine.

Connection/communication between HIL hardware and ECUs is implemented through busses and electricity, and it involves input/output operations. In today’s new modern machines the number of ECUs becomes higher and higher and the complexity increases with this. For example, in a car, such electronic features, as electronic fuel injection setup, braking system and others can be controlled by separate ECUs. Figure 11 shows appearance of ECU.

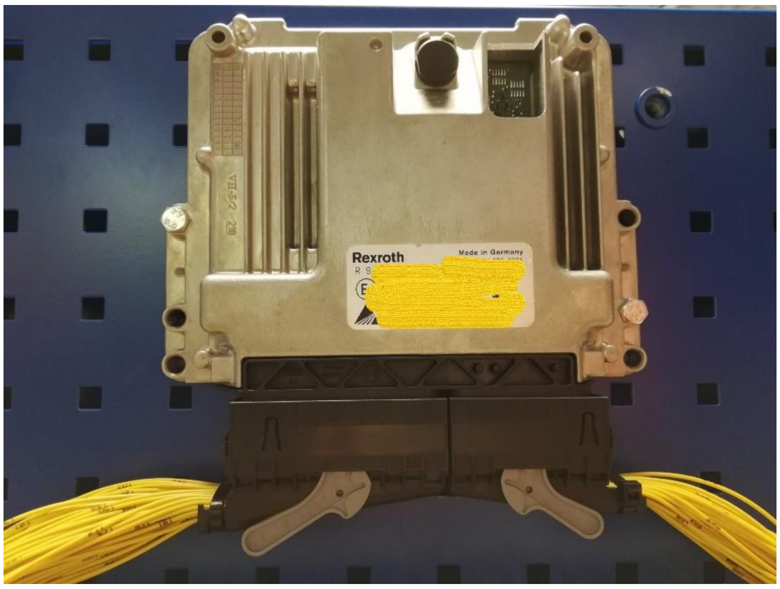


Figure 11. Example of ECU

An ECU fed with some number of inputs. Received from these inputs data can be e.g. a clutch pedal, accelerator position, or engine speed. The ECU processes the received data and alters the behaviour of the components of some system, such as ignition system and fuel injectors by sending control signal data by outputs.

6.4 Hardware

On the market, there are different providers of tools for developing, testing and calibrating electronic control units, e.g. dSPACE (digital signal processing and control engineering) or Speedgoat. The simulation system can consist of one or several cabinets, where equipment and wirings are installed. The equipment includes processor and I/O boxes, current to voltage converts, frequency converters, power source and PC. (HIL testing 2019.)

7 Design and technology choices

The main subject of this chapter is the presentation of the made decisions, justification of the selected technologies and the description of the implementation. As mentioned before, the main goal of this project was to find a solution for automatic creating test routines from a test drive test run, which could be implemented in the test automation process.

At first, the system as a whole is discussed. The development starts from the definition of requirements and conditions, planning of working process, and therefore the project management is handled in Polarion ALM. Jenkins is used for test execution. The test results can be provided by Polarion Jenkins Connector (see Figure 12).

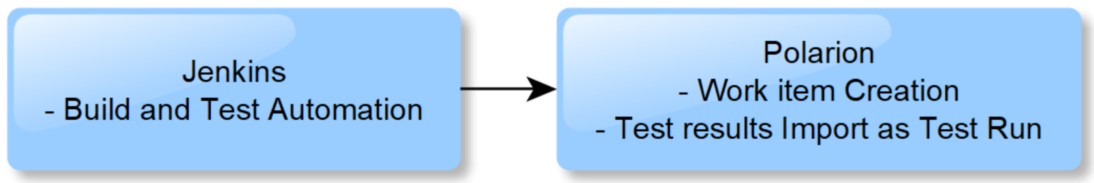

Jenkins
- Build and Test Automation
Polarion
- Work item Creation
- Test results Import as Test Run

Figure 12. Polarion Jenkins Connector main functions

Jenkins puts into operation the scheduled jobs and runs Robot Test Suites. Robot Framework provides the execution environment for automation test scripts. In addition, Jenkins updates the developed robot keyword library and robot scripts from SVN server. New Robot Scripts can be created writing manually or using Test Script Creation Software.

The tasks of CI platform will be:

- Retrieve the latest software versions from the version control system
- Download the software to the embedded hardware
- Retrieve and download the correct simulation model for the hardware
- Apply for the latest software test cases
- Perform and report tests

Figure 13 represents the architecture of the system as a whole. As can be seen, the results from the testing machine can be received through CANUSB and TAPI. TAPI should be created exactly for tested machine.

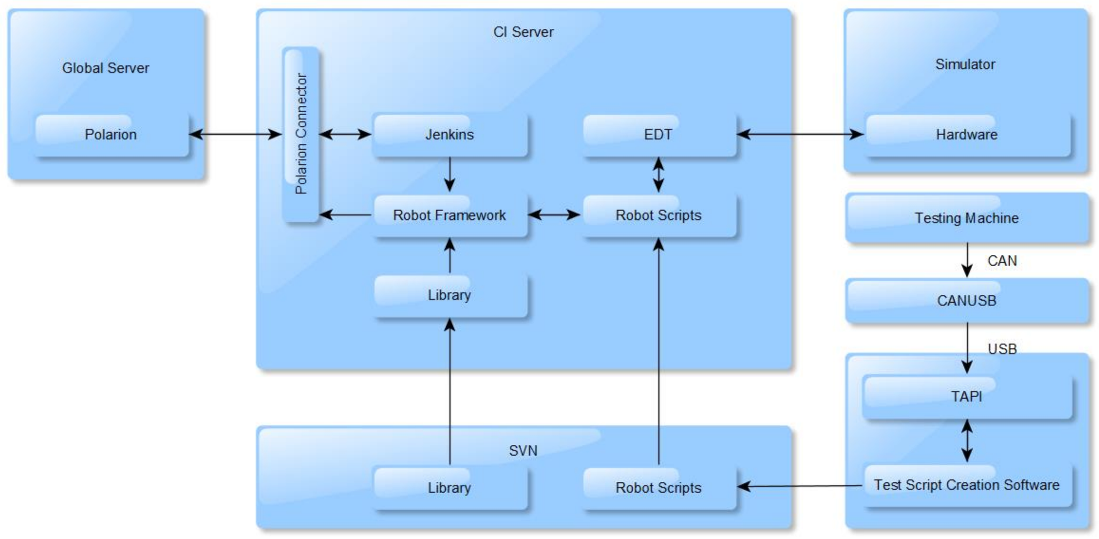


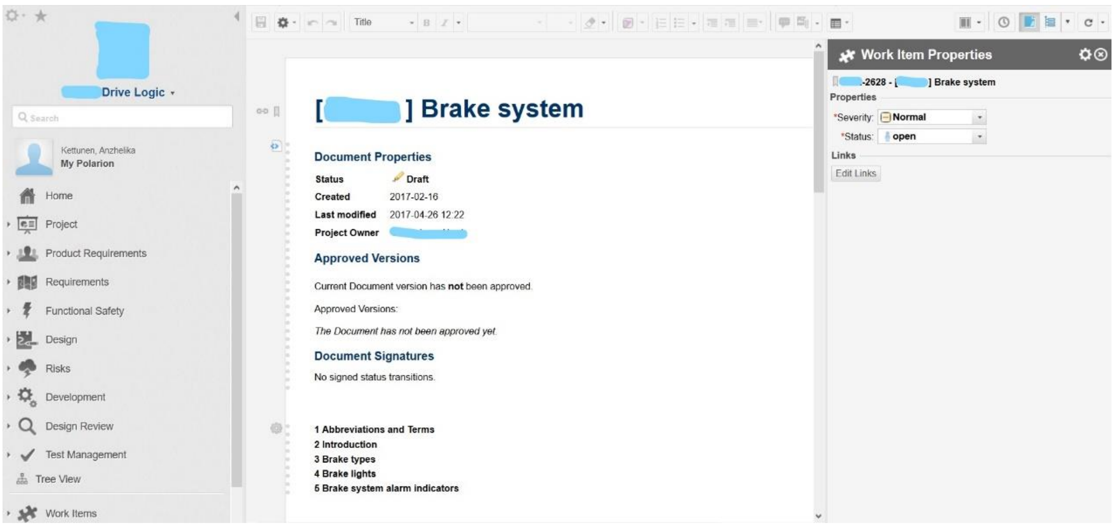
Figure 13. Test automation architecture

7.1 Polarion ALM

Polarion ALM is a consistent solution for software lifecycle management. Polarion provides traceability and transparency of software development and supports all the most important software development activities.

In 2005, Polarion company released the first 100% browser-based development platform designed to optimize enterprise development. In 2016, Siemens PLM Software acquired Polarion Software to expand their support for ALM market. During 10 years of growth, Polarion got over 2.5 million users. These numbers confirm that the users trust this platform. (About Polarion Software, a Siemens Company 2018.)

Polarion ALM is a very good solution in case of difficult environment and distributed documentation management. Data types in Polarion can be specified, for example requirement, test case, task, defect, and package. Figure 14 shows example document view in Polarion.



[redacted] Drive Logic

Kettunen, Anzhelika
My Polarion

[[redacted]] Brake system

Document Properties

Status	Draft
Created	2017-02-16
Last modified	2017-04-26 12:22
Project Owner	[redacted]

Approved Versions

Current Document version has **not** been approved.

Approved Versions:

The Document has not been approved yet.

Document Signatures

No signed status transitions.

1 Abbreviations and Terms
2 Introduction
3 Brake types
4 Brake lights
5 Brake system alarm indicators

Work Item Properties

[redacted]-2628 - [[redacted]] Brake system

Properties

*Severity:	Normal
*Status:	open

Links

Figure 14. Polarion screenshot

7.2 Robot Framework

Robot Framework is an open source software initially was developed at Nokia Networks. Nowadays it is sponsored by Robot Framework Foundation.

Robot Framework is application independent and provides easy-to-read result reports in HTML format, gives opportunities to create data-driven test cases, different levels setups and teardowns, and comes with its own reusable keywords.

Scripts can be saved into a file with .robot extension. Robot file contains the setting table, variables, test cases and keywords.

Setting table

Setting table contains the test case related settings such as tags, template, library, metadata, documentation, test and suite setup and teardown.

Tags are used for classifying test cases. There are two types of tags: (Robot Framework User Guide 2019)

- **Force Tags:** all test cases always get specified tags in a test case file with this setting.
- **Default Tags:** only those test cases get these tags which do not have a tag setting of their own.

Test templates convert test cases into data-driven tests. Such test cases contain only the arguments for the template keyword.

Library: The test library contains library keywords, which interact with the system under test. Remote library is used in a case when test libraries are located on different machines than where Robot Framework itself is running.

Test teardown executes after a test case or in situations if a test case fails.

Metadata is an information set shown in test reports and logs.

Variables mentioned in this section are necessary for settings, for example global variables, path to specific folder, address.

Test cases: The first column contains test case names, and the second column has keyword names.

Keywords: This part describes Suite Setup (executes once in the beginning), Test Setup (executes before every test case), Test Teardown (executes after every test case), and Suite Teardown (executes for the suite once in the end).

For example: a vehicle acceleration test. The steps are following:

1. Suite Setup: changing shuttle lever position to “forward”.
2. Test Case “Test Acceleration Pedal 100%” executing using arguments: percent of press (100) and expected speed (43).
3. Keyword “Test pedal”, test steps: acceleration pedal is pressed 100%, after 5 seconds the vehicle speed must be 43 km/h. If condition is not met, the test is failed.
4. Keyword “Teardown” executes (test teardown executes after every test case).
5. All following test cases will be executed the same way.
6. Suite Teardown: changing shuttle lever position to “forward”.

The file content is demonstrated below:

```
*** Settings ***
Library                 Remote    http://${ADDRESS}:${PORT}
Documentation           Example of robot file
Suite Setup             Setup
Test Teardown           Teardown
Suite Teardown          Suite Teardown

*** Variables ***
${ADDRESS}              %{ROBOT_SERVER_ADDRESS}
${PORT}                 8265

*** Test Cases ***
Test Acceleration Pedal 100%
    Test pedal      100    43

Test Acceleration Pedal 50%
    Test pedal      50     22
```

```
***Keywords ***
Setup
    Set Shuttle Lever  forward

Test pedal
    [Arguments]  ${accel_press}    ${speed}
    Set Acceleration Pedal   ${accel_press}
    Wait   5s
    Vehicle Speed Should Be Above   ${speed}

Teardown
    Set Acceleration Pedal  0
    Wait Until Vehicle Speed Is Below    0.1

Suite Teardown
    Set Shuttle Lever  park
```

Execution:

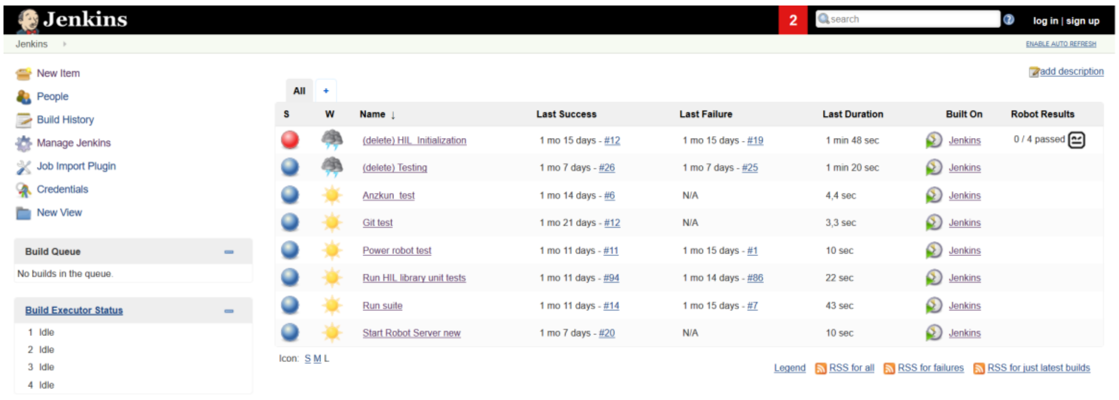
```
C:\Robot tests>robot "Example.robot"
```

7.3 Jenkins

Depending of users' needs, Jenkins can be used as a simple CI server or turned into a continuous delivery hub for any project. Configuration takes place via web interface.

The installation is quite simple; it was enough to download Jenkins from web-site, open up a terminal in the download directory and run provided Java script. After that, one can start to use Jenkins in the browser by opening http://localhost:8080.

Figure 15 demonstrates the view of main page.



Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle
3 Idle
4 Idle

S	W	Name ↓	Last Success	Last Failure	Last Duration	Built On	Robot Results
🔴	⛈	(delete) HIL_Initialization	1 mo 15 days - #12	1 mo 15 days - #19	1 min 48 sec	Jenkins	0 / 4 passed
🔵	⛈	(delete) Testing	1 mo 7 days - #26	1 mo 7 days - #25	1 min 20 sec	Jenkins	
🔵	☀	Anzkun_test	1 mo 14 days - #6	N/A	4,4 sec	Jenkins	
🔵	☀	Git test	1 mo 21 days - #12	N/A	3,3 sec	Jenkins	
🔵	☀	Power robot test	1 mo 11 days - #11	1 mo 15 days - #1	10 sec	Jenkins	
🔵	☀	Run HIL library unit tests	1 mo 11 days - #94	1 mo 14 days - #86	22 sec	Jenkins	
🔵	☀	Run suite	1 mo 11 days - #14	1 mo 15 days - #7	43 sec	Jenkins	
🔵	☀	Start Robot Server new	1 mo 7 days - #20	N/A	10 sec	Jenkins	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Figure 15. Jenkins screenshot

The list below demonstrates some of plugins, which can be useful in this project:

Git

Git is a distributed version control system for software projects allowing migration between different stages of development and keeping track of all application changes.

CVS

Jenkins can be integrated with CVS version control system using a modified version of the NetBeans CVS Client.

Build Pipeline

Build Pipeline plugin can be useful to define triggers for jobs that are required prior to execution.

Job Import Plugin

Gives opportunity to import jobs from another instance.

Parameterized Scheduler

This plugin can be used to configure a timer schedule for parameterized builds. For example, we can configure schedule so, that it will start builds every night. It is quite useful if during daytime developers make changes in a code, and the tests will be run at night time.

PostBuildScript plugin

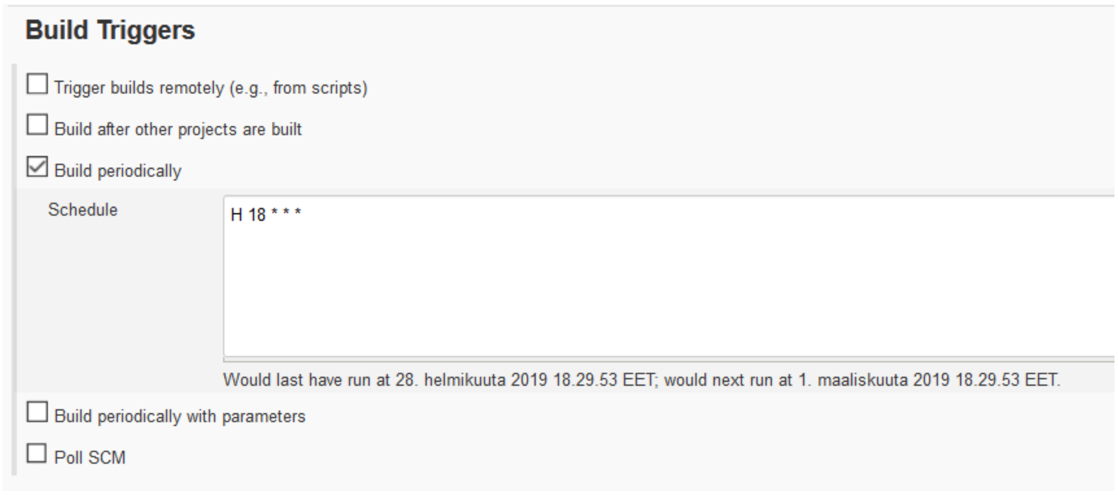
PostBuildScript plugin can run one or more configurable actions after a build, which depends e.g. on the build result.

Robot Framework plugin

This plugin allows to store Robot Framework test reports and show the summaries in a project.

Build Triggers

Figure 16 shows an example of the usage of build triggers. In this example the schedule for execution is set so that the run will start every day between 18:00 and 19:00.


Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [x] Build periodically

Schedule

```
H 18 * * *
```

Would last have run at 28. helmikuuta 2019 18.29.53 EET; would next run at 1. maaliskuuta 2019 18.29.53 EET.

- [ ] Build periodically with parameters
- [ ] Poll SCM

Figure 16. Build Triggers

7.4 Communication flow

Web socket communication

Figure 17 describes web socket communication between client and server. Client sends handshake to the server and starts waiting for handshake response. After response is received, client sends request and starts waiting for the next response and results. At this stage when data is received, the connection can be closed (timeout can be the reason of connection close as well). Client sends close connection request, and connection will be closed immediately after confirmation has been received or due to the expiration of time.

The server is in standby mode and waiting for a request from client. When handshake request is received, server responses to it and starts waiting for a request. After request is received, server sends response for request, executes request and sends results for request. Connection will be closed after close connection request received or by the expiration of time.

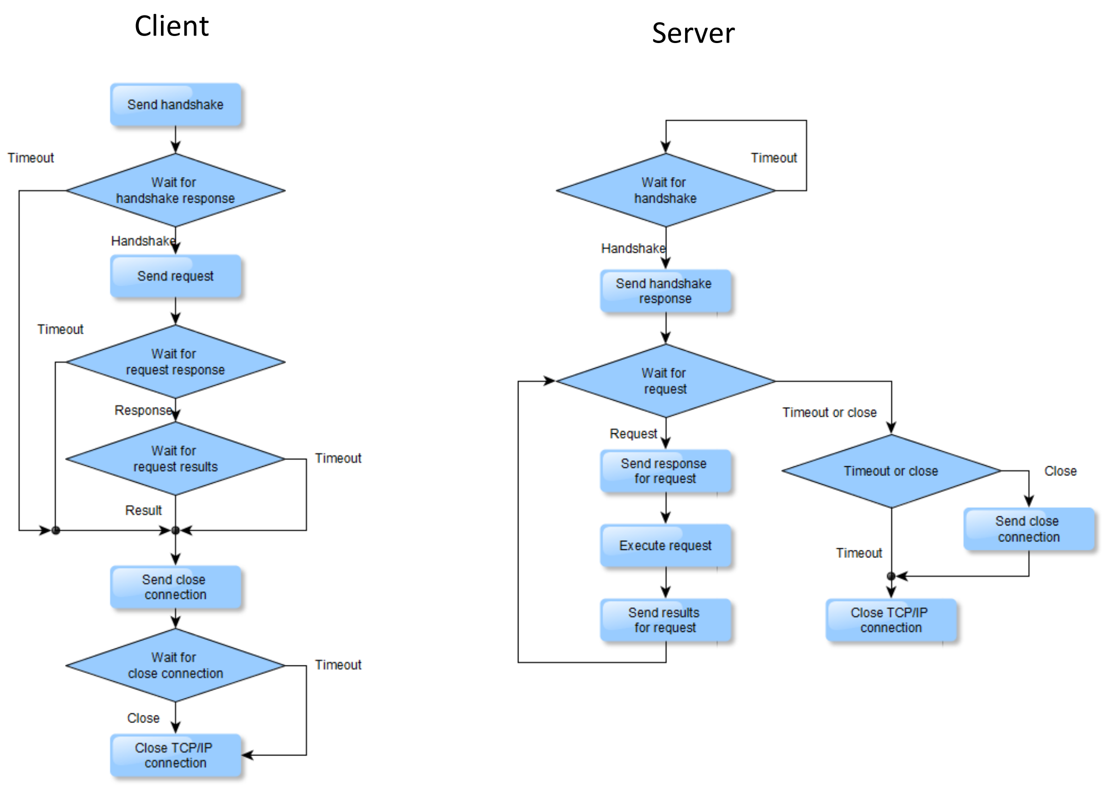


Figure 17. Web Socket Communication

Test flow

The steps of communication flow in the system between ALM, CI and HIL are described below:

1. Polarion contains requirements and test cases, which refer to robot scripts by identification number.
2. Source code can be stored on SVN server:
    a. HILLibrary (keywords, variables)
    b. Unit tests
    c. Robot scripts
    d. Jenkins properties (robot arguments and variables, paths, robot server address etc.)
3. Jenkins updates changes from SVN server and executes the Robot Framework scripts according to the schedule.
4. Robot Framework runs a test script.
5. The test script opens a web socket connection to the WSS (Web Socket Server).
6. The WSS connects to the correct HIL API.
7. Robot Framework provides test results to Polarion connector in XML mode.
8. Polarion connector writes test results to Polarion.

7.5 TAPI

API, Application Programming Interface is a programming interface that consists of a set of communication protocols, tools and implementation methods, and defines the means of application communication between applications. In this project test API is used for getting real machine signals through CANUSB, which connects real machine under manual test with PC (script creation software is installed on it). TAPI is implemented as dynamic-link library (or DLL). Script creation software communicates with it simply with functions and arguments. It could be imaged that TAPI generates the table of signals containing the values for each of them based on CAN messages, which are coming periodically. TAPI decodes the messages, saves signal values in a table and provides them to script creation software by request.

8 Script Creation Software (development result)

8.1 Workflow

Previous chapters described ready systems already available for use. The purpose of this chapter is to describe the script creation software's principles, concept and logic, which were developed 100% inside this project. The main goal of this application is to generate ready robot test scripts based on manual machine tests. The demo version demonstrates testing capabilities and basic requirements on the example of implementation and usage script creation software in heavy moving machinery HIL testing. Visual Studio along with C# was used to create a Windows Forms demo application. The purpose of this application was to define the necessary data and parameters to make script creation possible based on signal changing.

The basic principle of application's work is following:

- Receiving signals from the machine during manual test
- Filtering of signals
- Recording received data
- Data analysing
- Robot script generation

Figure 18 demonstrates the stages of script creation workflow from the determination of the requirements for the test machine till the delivery of ready scripts to SVN.

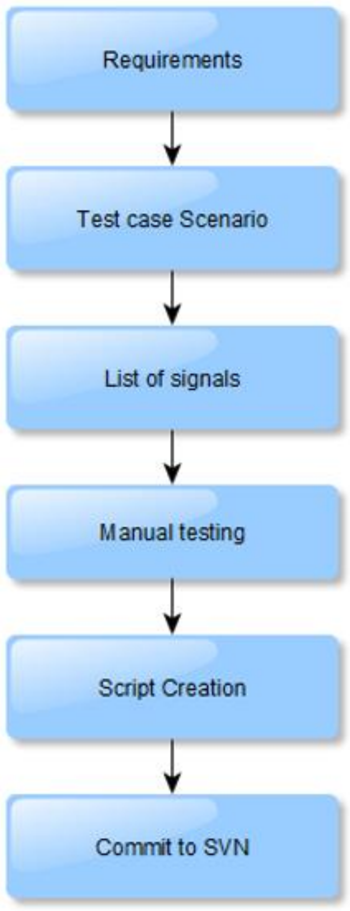


Figure 18. Script creation workflow

8.2 Requirements

Any testing is performed based on requirements, how testing machine or some other technical device should respond to external influences. In case of moving machines, the example of requirements can be: “Parking brake indicator is solid ON when parking brake is engaged”, “Parking brake engagement is allowed only when the speed of the machine is below speed limit parameter of parking brake”.

8.3 Test case scenario

Based on the requirements, the test case scenario is created. Usually it consists of preconditions, test steps and expected results. The test case scenario provides the sequence of actions, how the machine should be tested and which results should be received to define that test was passed successfully.

The following is an example of parking brake status and parking brake indicator testing. Table 1 represents the preconditions for manual test, which must be met before the first test action.

Table 1. Preconditions for manual test

Preconditions
Engine running
Operator seated
None of the pedals are pressed
Shuttle lever is in P position

Often preconditions match the default conditions for HIL testing, however, in case some of them differ, they must be mentioned obligatorily and performed as steps of the test case.

The next table demonstrates an example of test case steps and the expected results for them.

Table 2. Test case steps and expected results

Step	**Expected Result**
Set Shuttle Lever to R	Parking Brake Indicator is OFF Machine speed increases over 5 km/h
Press Clatch and Brake pedals	Machine stops
Set Shuttle Lever to F and release pedals	

Press Acceleration pedal on 50%	Machine speed increases over 10 km/h
Release Acceleration pedal	
Set Shuttle Lever to P	Parking Brake Indicator is ON

This means that the test operator during testing performs the provided actions, checks results of actions and compares them with the expected results of the test case. In case the results differ from the expected ones, the test is considered failed

8.4 List of signals

In case of a complicated system, a machine can provide quite a large amount of signals, hundreds or even thousands. Usually therequirement means testing only a few of the signals, which is why the filtering of signals should be implemented; otherwise the application provides a robot script with a large amount of conditions, which would test the whole system. It increases the probability to get a robot test, which will fail in an HIL system, and a failure can occur for any other reason, not only due to the tested conditions.

The list of signals needed for a particular test must also provide information about every signal:

- Input/output signal flag
- Script template
- Conditions for script creation
- Default value

8.4.1 Input/output signal flag

Script is generated based on the detection of the change in signal value. If the value of some signal is different than it was before, the script will be generated based on the script template and script conditions. In case if in the moment of comparing signals was detected that more than one of the signals is changed, the script for input signal will be generated first.

For example, shuttle lever status has input flag, parking brake status has output flag. If during a manual test it was detected that at the same time the shuttle lever status and parking brake status values are different from the values received before it, at first the script for shuttle lever state must be generated, because the input signal produces an impact on the output signal. It will be correct that on HIL system shuttle lever state should be changed first, and only after that the output signal for parking brake state should be checked.

8.4.2 Script template

Script template represents the HIL library keyword, which should be used for script generation.

For example: output signal ParkingBrake could have keyword “Parking Brake Status Should Be”. During testing the operator checks parking brake status and compares it with a given expected result from a test case. However, in the process of robot script generation based on the recorded signals from a real machine, the application perceives the received signals’ values as required and uses them in the scripts.

8.4.3 Conditions for script creation

There are different types of signals, some of them with only two values, 0 and 1, “off“ and “on”. Other signals could have several possible values (shuttle lever status: park, neutral, forward, reverse) or even floating values (vehicle speed, pedal position). It is quite important to use detailed conditions for signals, which have floating values.

Example 1: signal “ParkingBrake” can have only “0” or “1” values. “0” means, that parking brake is “off”, and “1” means parking brake is “on”. In such cases two cases must be mentioned:

- case=0:off
- case=1:on

In this demo version following syntax was used:

case<comparison sign>:<output for script>

Comparison sign could be “<”, “>” or “=”.

Hence, if during the analysis of the recorded signals is detected that “ParkingBrake” signal was changed from 1 to 0, the generated script will be “Parking Brake State Should Be   off”.

Example 2: signal “ShuttleLeverStatus” can have several values, so cases could be mentioned in the same way like in previous example:

- case=4:park
- case=1:neutral
- case=3:reverse
- case=2:forward

Example 3: signal “VehicleSpeed” has float values, there is no need to generate script every time when change was detected. For such cases logic of script generation was developed for floating values. At first it should be decided, which values are the most interesting in this test. The chapter “Test case scenario” provided an example, where the vehicle speed points can be noticed which should be noticed during the test: “Machine speed increases over 5 km/h” and “Machine speed increases over 10 km/h”. Situations, when speed changes and goes over 5 km/h and over 10 km/h are of interest; also it could be useful to detect, when the machine stops, so the speed drops under 1 km/h. The script template for “VehicleSpeed” is “Wait Until Value Is”. The conditions could be:

- case>5: Above CANSpeed 5 (Wait Until Value Is Above CANSpeed 5)
- case>10:Above CANSpeed 10 (Wait Until Value Is Above CANSpeed 10)
- case<1:Below CANSpeed 1 (Wait Until Value Is Below   CANSpeed   1)

The demo version of the product uses conditions saved in .txt file. It is recommendable to save the data in some other format.

8.4.4 Default value

Default values should be used for teardown generation, also they could be useful in the beginning of an analysis for comparing the first received values with default values.

Examples:

- Input signal “ShuttleLever” def=4
  - case=4:park, teardown script will be “Set Shuttle Lever   park”
- Output signal “ParkingBrake” def=1

- case=1:on, so default state for parking brake is “on”, teardown script will be “Parking Brake Status Should Be   on”

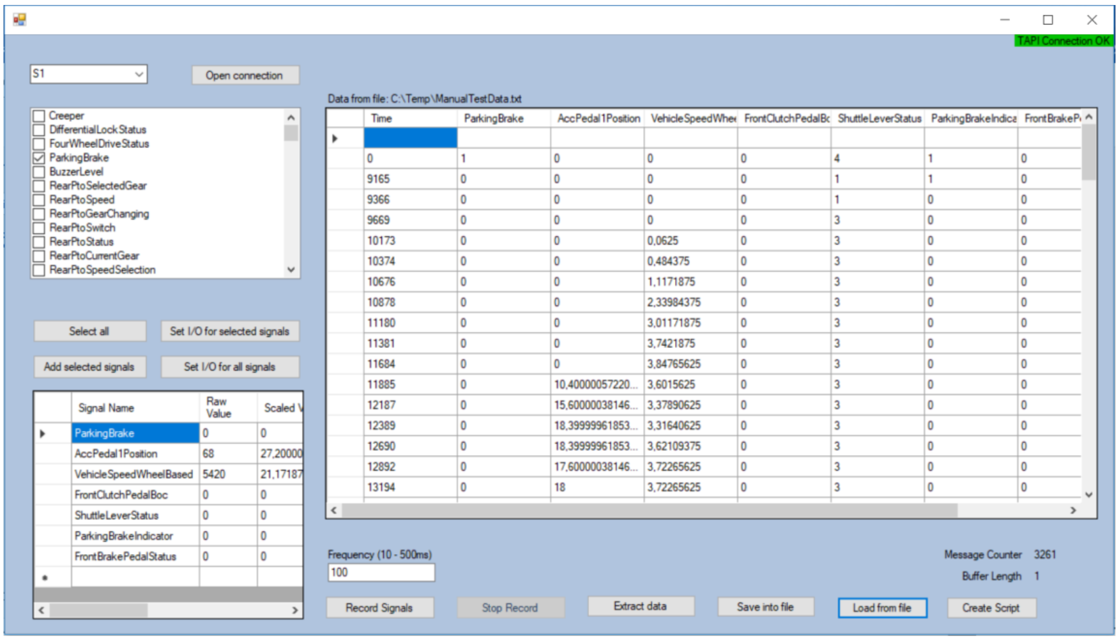
8.5 Manual testing

The demo version of developed application is demonstrated in this chapter.

Work flow:

1. Operator connects the computer to the test machine using CANUSB.
2. Launchs script creation software before manual test started.
3. Executes preconditions, mentioned in test case.
4. Using script creation SW operator starts recording of signals.
5. After manual test case is finished, operator stops recording.
6. If needed, received data can be saved into file.
7. Script creation SW generates test scripts and save them into robot file.

Figure 19 demonstrates the main view of the developed application. The operator can choose the model of the tested machine (in this case it is “S1”). After it, TAPI connection must be established (button “Open connection”). The application provides available signals for the selected model. Here the signals needed for upcoming test should be marked.



☐ Creeper
☐ DifferentialLockStatus
☐ FourWheelDriveStatus
☑ ParkingBrake
☐ BuzzerLevel
☐ RearPtoSelectedGear
☐ RearPtoSpeed
☐ RearPtoGearChanging
☐ RearPtoSwitch
☐ RearPtoStatus
☐ RearPtoCurrentGear
☐ RearPtoSpeedSelection

Signal Name	Raw Value	Scaled V
ParkingBrake	0	0
AccPedal1Position	68	27,20000
VehicleSpeedWheelBased	5420	21,17187
FrontClutchPedalBoc	0	0
ShuttleLeverStatus	0	0
ParkingBrakeIndicator	0	0
FrontBrakePedalStatus	0	0

Data from file: C:\Temp\ManualTestData.txt

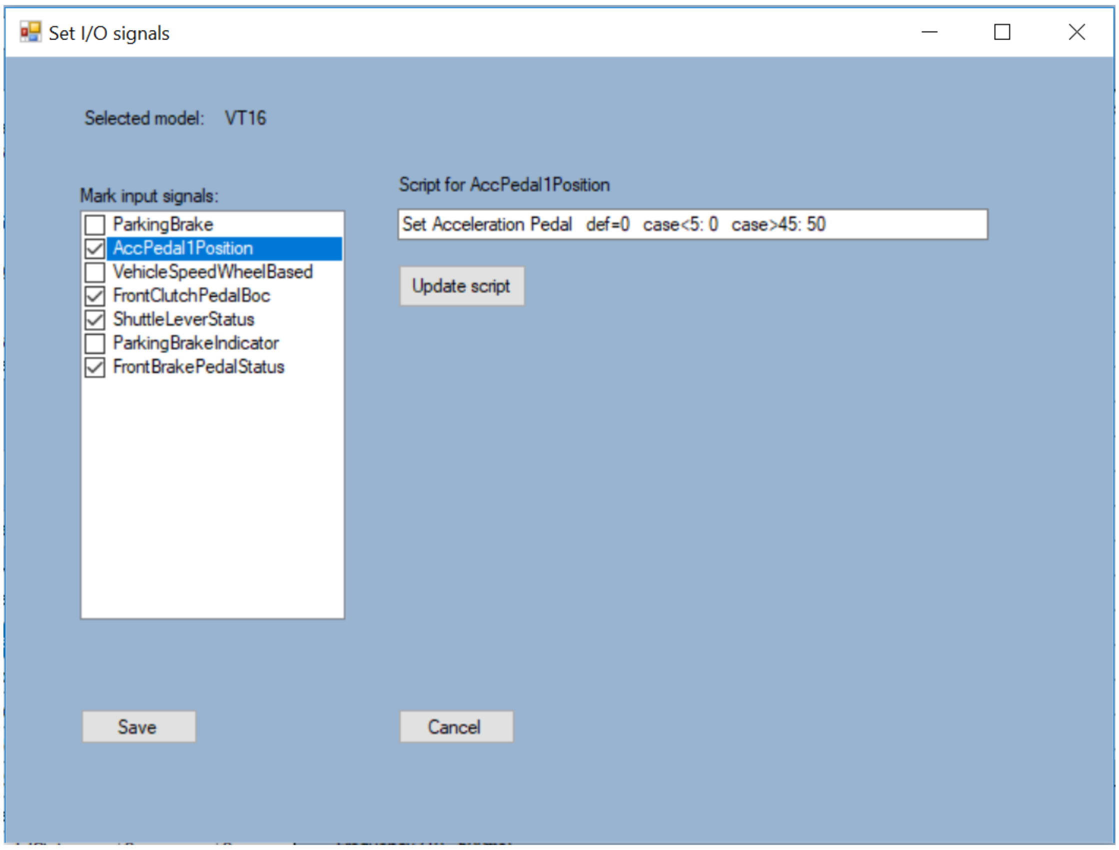
Time	ParkingBrake	AccPedal1Position	VehicleSpeedWhe	FrontClutchPedalBo	ShuttleLeverStatus	ParkingBrakeIndica	FrontBrakeP
0	1	0	0	0	4	1	0
9165	0	0	0	0	1	1	0
9366	0	0	0	0	1	0	0
9669	0	0	0	0	3	0	0
10173	0	0	0,0625	0	3	0	0
10374	0	0	0,484375	0	3	0	0
10676	0	0	1,1171875	0	3	0	0
10878	0	0	2,33984375	0	3	0	0
11180	0	0	3,01171875	0	3	0	0
11381	0	0	3,7421875	0	3	0	0
11684	0	0	3,84765625	0	3	0	0
11885	0	10,40000057220...	3,6015625	0	3	0	0
12187	0	15,60000038146...	3,37890625	0	3	0	0
12389	0	18,39999961853...	3,31640625	0	3	0	0
12690	0	18,39999961853...	3,62109375	0	3	0	0
12892	0	17,60000038146...	3,72265625	0	3	0	0
13194	0	18	3,72265625	0	3	0	0

Frequency (10 - 500ms): 100

Message Counter 3261
Buffer Length 1

Figure 19. Main form of application

Pressing button “Set I/O for selected signals” opens a form, which provides the opportunity to modify the input/output signal flag, script template, conditions for script creation and default value (see chapter 8.3 List of signals). Form “Set I/O signals” is demonstrated in Figure 20.



Selected model: VT16

Mark input signals:

- [ ] ParkingBrake
- [✓] AccPedal1Position
- [ ] VehicleSpeedWheelBased
- [✓] FrontClutchPedalBoc
- [✓] ShuttleLeverStatus
- [ ] ParkingBrakeIndicator
- [✓] FrontBrakePedalStatus

Script for AccPedal1Position

Set Acceleration Pedal def=0 case<5: 0 case>45: 50

Figure 20. Form "Set I/O signals"

After modifications are made and “Save” button pressed, the file containing signal data for the selected machine model will be updated.

In the main application form, the operator presses the button “Add selected signals” and can see the current signal values for each signal in DataGridView in the lower left corner of the form containing signal name, raw value, scaled value and some other information.

The operator starts recording the signals and executes the actions described in the test case. When the manual test is finished, the operator stops recording. After that, he/she can see the table of recorded values for each signal. The application checks and records signals every 100 ms (frequency can be changed) even despite the fact that no signal was changed from the last record, it causes duplicating of rows, where only “Time” column has a different value. Button “Extract” helps to clear the table by deleting duplicate rows. The recorded signals can be saved to a file or loaded from a file created before.

At this stage, the operator can press “Create script” button, choose a file name and location for the robot test file, after which the script will be generated and saved into the mentioned place.

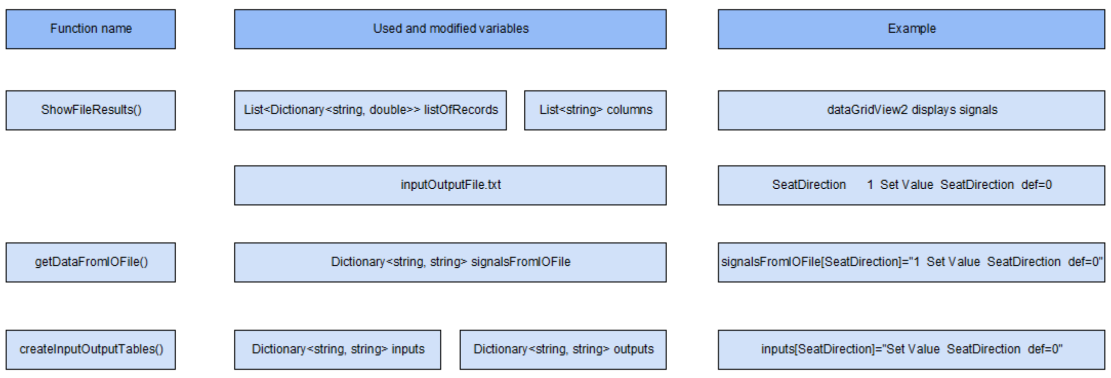
8.6 Script creation

The generated script will be saved in the robot file. The logic of script creation is demonstrated in Appendixes 5-7. After “Create script” button was clicked, Save-FileDialog requests to mention the file name for the robot file. As script templates and other necessary information for each signal are saved in the file, the application tries to get data from input/output file and fill in two tables:

```
// Input table
private  Dictionary<string, string> inputs = new Dictionary<string, string>();
// Output table
private  Dictionary<string, string> outputs = new Dictionary<string, string>();
```

The tables contain only the signals meant for the current test case.

Figure 21 demonstrates some of the functions and variables used or modified in the mentioned function, and gives an example, what kind of data the variable contains. This information can be useful in case of deep application code analysis, which is performed in appendixes 1-9.



Function name	Used and modified variables		Example
ShowFileResults()	List<Dictionary<string, double>> listOfRecords	List<string> columns	dataGridView2 displays signals
	inputOutputFile.txt		SeatDirection 1 Set Value SeatDirection def=0
getDataFromIOFile()	Dictionary<string, string> signalsFromIOFile		signalsFromIOFile[SeatDirection]="1 Set Value SeatDirection def=0"
createInputOutputTables()	Dictionary<string, string> inputs	Dictionary<string, string> outputs	inputs[SeatDirection]="Set Value SeatDirection def=0"

Figure 21. Examples of data

ShowFileResults() function is used to display just the recorded or loaded data from the file. Functions getDataFromIOFile() and createInputOutputTables() are used for script creation.

After “inputs” and “outputs” tables are filled, the application goes through each key-value pair in “inputs” table, gets default value for signal and compares it with the value in the first row of records. If the value is not the same as the default, the script will be generated based on the template for the current signal. When all input signals have been compared, the same operation is repeated for the output signals.

Script creation diagram in Appendix 6 demonstrates the next step, the current signal value compared with the previous value. The loop is used to go from the second until the last row of table (list of records). The same way, the input signals are checked first and compared with the previous values for the same signal. If a value is changed, a script will be added. After the input signals the output signals will be compared.

There can be special cases, such as for signal “VehicleSpeedWheelBased”, which has the script template “Wait Until Value Is”. If at the same time more than one signal was changed, the script “Wait Until Value Is” must appear before others.

Appendix 7 represents the final part, teardown generation. The same way as the first input signals are checked, the generated scripts set the default values for them, after which the teardown for output signals will be done.

After all data is save into the file, it will be opened for the operator.

8.7 Commit to SVN

A generated robot file must be committed to SVN, so it can be used for HIL testing by CI platform.

9 Developed software testing

The developed application was successfully tested on a real machine. The stages from requirements specification until commitment to SVN are described in the re-port. In this chapter, the real test is presented, which confirms the correctness of the generated scripts.

Test case

The test case can be found in chapter 8.3 Test case scenario.

List of signals

The list of signals is saved in .txt file:

```
ParkingBrake                  0  Parking Brake Status Should Be   def=1   case=0:off   case=1:on

AccPedal1Position             1  Set Acceleration Pedal   def=0   case<5:0   case>45:50

VehicleSpeedWheelBased        0  Wait Until Value Is   case>5:Above  CANSpeed  5   case>10:Above
CANSpeed  10   def=0,1   case<1:Below  CANSpeed  1

ShuttleLeverStatus            1  Set Shuttle Lever   def=4   case=4:park   case=1:neutral   case=3:reverse
case=2:forward

ParkingBrakeIndicator         0  Parking Brake Indicator State Should Be   def=1   case=0:off   case=1:on

FrontBrakePedalStatus         1  Press Brake   def=0   case=0:0   case=2:50

FrontClutchPedalBoc           1  Set Clutch Pedal   def=0   case=0:0   case=1:100
```

Data was used this way (in txt file) only by the reason of limited development time. The best way to save and use such kind of data, using database.

Manual test vs. generated script

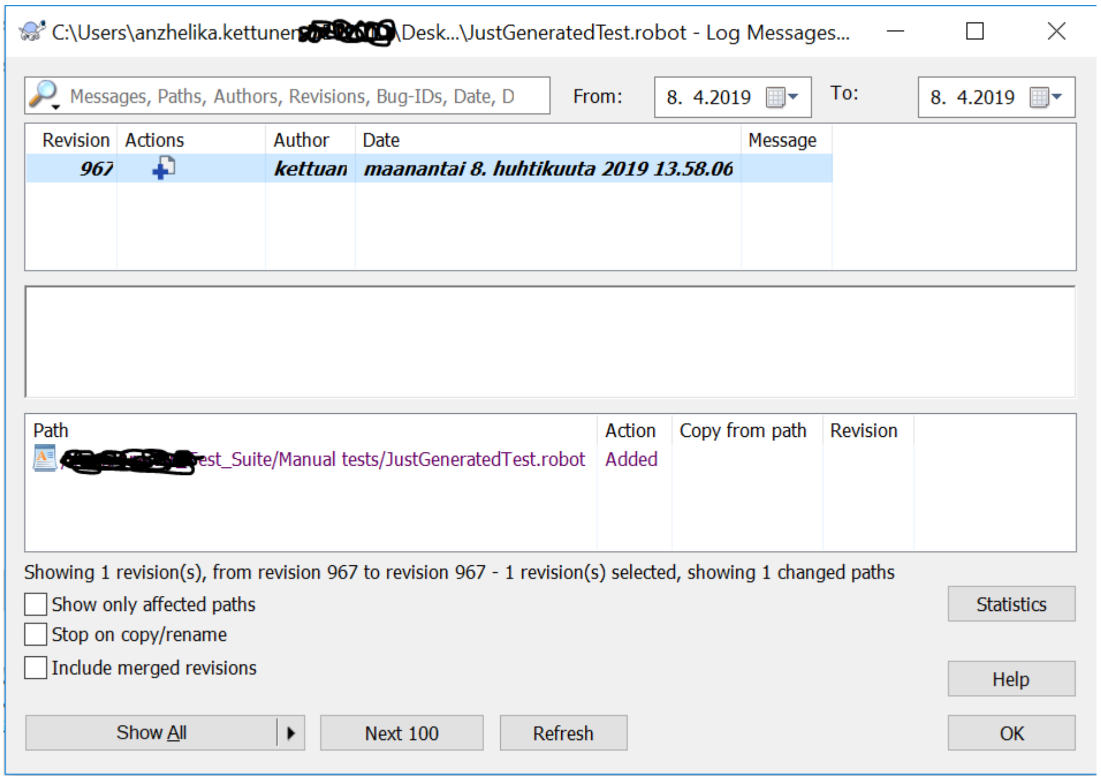
Appendix 10 represents the comparison of manual test actions with the generated script. As can be seen, the generated robot test has more actions than was described in the manual test. For example: in manual test one step “Set Shuttle Lever reverse position” generated four scripts for HIL testing:

- Set Shuttle Lever neutral
- Parking Brake Status Should Be off
- Parking Brake Indicator State Should Be off
- Set Shuttle Lever reverse

In the tested machine the shuttle lever is by default in park position. When the operator changes the position from park to reverse, the shuttle lever goes through neutral position, which is the reason why the application generated it. This is quite good for HIL testing, because the simulator must work exactly the same way like a real machine. At the same time it will be tested that already at this stage the parking brake status and parking brake indicator state must be changed, not only after shuttle lever is set to reverse position as described in the manual test. Therefore, the generated script is much more accurate than if it could be written manually.

Commit to SVN

For the test purposes the folder named “Manual tests” was created. The robot file generated by the test application was saved into it and committed to SVN as seen in Figure 22.



C:\Users\anzhelika.kettuner[redacted]\Desk...\JustGeneratedTest.robot - Log Messages...

Messages, Paths, Authors, Revisions, Bug-IDs, Date, D From: 8. 4.2019 To: 8. 4.2019

Revision	Actions	Author	Date	Message
967	✚	kettuan	maanantai 8. huhtikuuta 2019 13.58.06	

Path	Action	Copy from path	Revision
[redacted]est_Suite/Manual tests/JustGeneratedTest.robot	Added		

Showing 1 revision(s), from revision 967 to revision 967 - 1 revision(s) selected, showing 1 changed paths

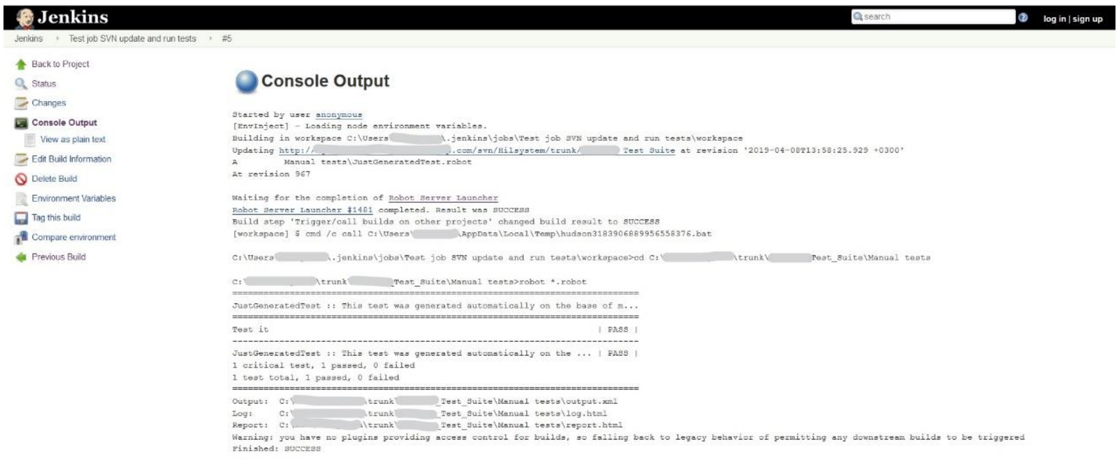
☐ Show only affected paths
☐ Stop on copy/rename
☐ Include merged revisions

Figure 22. Commit to SVN

Jenkins (CI platform)

Jenkins job was created to automate robot tests on HIL simulator. It can be scheduled to run e.g. every night. Figure 23 represents console output that happened during its run:

1. SVN updates were made
2. Robot Server was launched
3. Run all robot tests in folder “Manual tests” was executed.

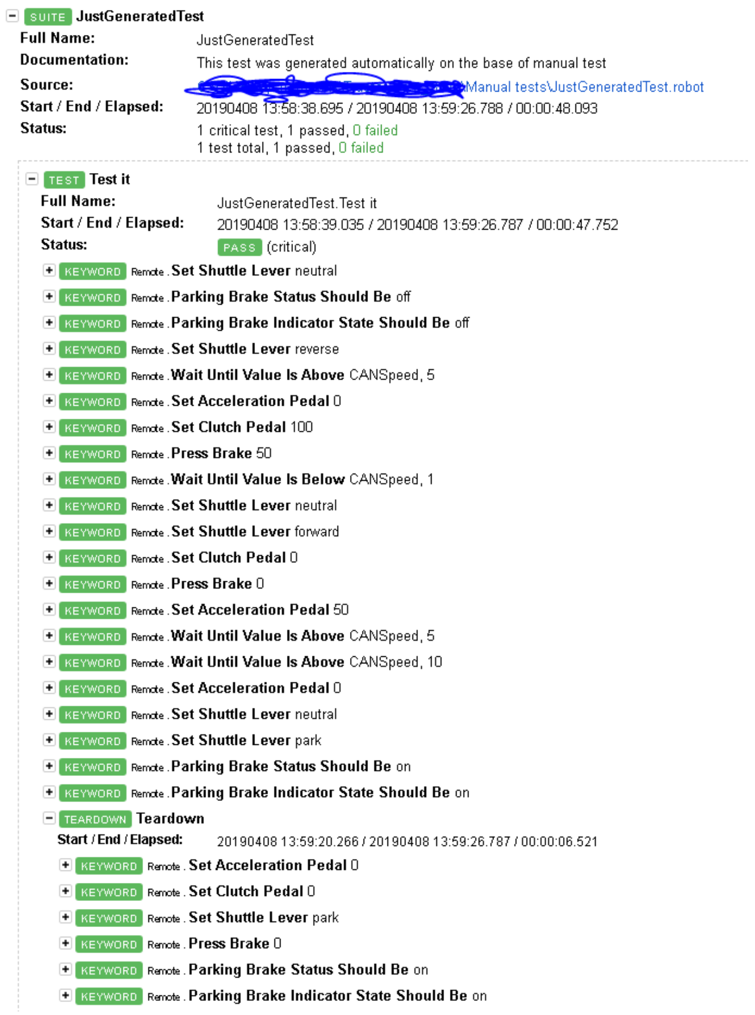


```
Started by user anonymous
[EnvInject] - Loading node environment variables.
Building in workspace C:\Users[redacted]\.jenkins\jobs\Test job SVN update and run tests\workspace
Updating http://[redacted].com/svn/Hilsystem/trunk/[redacted] Test Suite at revision '2019-04-08T13:58:25.929 +0300'
A         Manual tests\JustGeneratedTest.robot
At revision 967

Waiting for the completion of Robot Server Launcher
Robot Server Launcher #1481 completed. Result was SUCCESS
Build step 'Trigger/call builds on other projects' changed build result to SUCCESS
[workspace] $ cmd /c call C:\Users[redacted]\AppData\Local\Temp\hudson3183906889956558376.bat

C:\Users[redacted].jenkins\jobs\Test job SVN update and run tests\workspace>cd C:\[redacted]\trunk\[redacted]Test_Suite\Manual tests

C:\[redacted]\trunk\[redacted]Test_Suite\Manual tests>robot *.robot
==============================================================================
JustGeneratedTest :: This test was generated automatically on the base of m...
==============================================================================
Test it                                                               | PASS |
------------------------------------------------------------------------------
JustGeneratedTest :: This test was generated automatically on the ... | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\[redacted]\trunk\[redacted]_Test_Suite\Manual tests\output.xml
Log:     C:\[redacted]\trunk\[redacted]_Test_Suite\Manual tests\log.html
Report:  C:\[redacted]\trunk\[redacted]_Test_Suite\Manual tests\report.html
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of permitting any downstream builds to be triggered
Finished: SUCCESS
```

Figure 23. Jenkins console output

Test results

Jenkins created a test results report (report.html, the path of which can be found in the console output), shown in Figure 24.



```
⊟ SUITE JustGeneratedTest
  Full Name:            JustGeneratedTest
  Documentation:        This test was generated automatically on the base of manual test
  Source:               [redacted]Manual tests\JustGeneratedTest.robot
  Start / End / Elapsed: 20190408 13:58:38.695 / 20190408 13:59:26.788 / 00:00:48.093
  Status:               1 critical test, 1 passed, 0 failed
                        1 test total, 1 passed, 0 failed

  ⊟ TEST Test it
    Full Name:            JustGeneratedTest.Test it
    Start / End / Elapsed: 20190408 13:58:39.035 / 20190408 13:59:26.787 / 00:00:47.752
    Status:               PASS (critical)
    ⊞ KEYWORD Remote . Set Shuttle Lever neutral
    ⊞ KEYWORD Remote . Parking Brake Status Should Be off
    ⊞ KEYWORD Remote . Parking Brake Indicator State Should Be off
    ⊞ KEYWORD Remote . Set Shuttle Lever reverse
    ⊞ KEYWORD Remote . Wait Until Value Is Above CANSpeed, 5
    ⊞ KEYWORD Remote . Set Acceleration Pedal 0
    ⊞ KEYWORD Remote . Set Clutch Pedal 100
    ⊞ KEYWORD Remote . Press Brake 50
    ⊞ KEYWORD Remote . Wait Until Value Is Below CANSpeed, 1
    ⊞ KEYWORD Remote . Set Shuttle Lever neutral
    ⊞ KEYWORD Remote . Set Shuttle Lever forward
    ⊞ KEYWORD Remote . Set Clutch Pedal 0
    ⊞ KEYWORD Remote . Press Brake 0
    ⊞ KEYWORD Remote . Set Acceleration Pedal 50
    ⊞ KEYWORD Remote . Wait Until Value Is Above CANSpeed, 5
    ⊞ KEYWORD Remote . Wait Until Value Is Above CANSpeed, 10
    ⊞ KEYWORD Remote . Set Acceleration Pedal 0
    ⊞ KEYWORD Remote . Set Shuttle Lever neutral
    ⊞ KEYWORD Remote . Set Shuttle Lever park
    ⊞ KEYWORD Remote . Parking Brake Status Should Be on
    ⊞ KEYWORD Remote . Parking Brake Indicator State Should Be on
    ⊟ TEARDOWN Teardown
      Start / End / Elapsed:  20190408 13:59:20.266 / 20190408 13:59:26.787 / 00:00:06.521
      ⊞ KEYWORD Remote . Set Acceleration Pedal 0
      ⊞ KEYWORD Remote . Set Clutch Pedal 0
      ⊞ KEYWORD Remote . Set Shuttle Lever park
      ⊞ KEYWORD Remote . Press Brake 0
      ⊞ KEYWORD Remote . Parking Brake Status Should Be on
      ⊞ KEYWORD Remote . Parking Brake Indicator State Should Be on
```

Figure 24. Test results

As can be seen, the generated robot test passed successfully. The generated script can be found in Appendix 11.

10 Conclusions

As a result, an application was developed that creates robot test scripts based on a manual test. The most important issue in this project is conception. One of the goals was to get a confirmation that the implementation is possible even for quite complicated systems and to find out the script creation logic. The purpose was also to understand what data is needed and what is the best way to retrieve/update it, and in which form.

For practical usage this application could be developed with web-based user interface. The database should be taken into use for saving and retreaving information mentioned in chapter 8.4 “List of signals”, hence, it will contain:

- Basic information about machine models and list of available signals for each of them.
- Types of signals.
- Script templates (depend on test case).
- Conditions for script creation (also depend on test case).
- Default values.

The developed application allows to see the real machines’ signals during manual test execution, to save the signals’ values and generate robot scripts based on them. The application was also successfully tested in a real environment. The test results are demonstrated in this document in chapter 9 “Developed software testing”. The pre-set requirements and goals for the thesis were achieved, which means that the thesis project can be considered to be successful.

References

About Polarion Software, a Siemens Company. 2018. Siemens. Accessed on 11 March 2019. Retrieved from https://polarion.plm.automation.siemens.com/company/index

Application Lifecycle Management Software. 2019. Capterra. Accessed on 7 March 2019. Retrieved from https://www.capterra.com/application-lifecycle-management-software/

Buildbot Basics. N.d. Buldbot official web-site. Accessed on 29 March2019. Retrieved from https://buildbot.net/

Citrus Integration Testing. 2019. Citrus web-pages. Accessed on 28 February 2019. Retrieved from https://citrusframework.org/

Continuous integration. 2019. Wikipedia, the free encyclopedia. Accessed on 29 March 2019. Retrieved from https://en.wikipedia.org/wiki/Continuous_integration

CruiseControl. 2019. CruiseControl official web-pages. Accessed on 4 March 2019. Retrieved from http://cruisecontrol.sourceforge.net/

Cypress. 2019. Cypress web-pages. Accessed on 28 February 2019. Retrieved from https://www.cypress.io/

Devecto Oy. 2019. Devecto's web-pages. Accessed on 10 April 2019. Retrieved from https://devecto.com/

Electronic control unit. 2019. Wikipedia, the free encyclopedia. Accessed on 29 March 2019. Retrieved from https://en.wikipedia.org/wiki/Electronic_control_unit

HIL testing. 2019. dSPACE GmbH. Accessed on 29 March 2019. Retrieved from https://www.dspace.com/en/pub/home/products/systems/ecutest.cfm

Jenkins User Documentation. 2019. Jenkins. Accessed on 4 March 2019. Retrieved from https://jenkins.io/doc/

PJ – Polarion Jenkins Connector. 2016. Polarion Extention Portal. Accessed on 10 March 2019. Retrieved from http://extensions.polarion.com/extensions/171-pj-polarion-jenkins-connector

Polarion ALM. 2019. Siemens. Accessed on 29 March 2019. Retrieved from https://polarion.plm.automation.siemens.com/products/polarion-alm

Polarion Extentions. 2016. Polarion Extention Portal. Accessed on 5 March 2019. Retrieved from http://extensions.polarion.com

RD-hankkeet ja simulointi. N.d. Protacon's web-page. Accessed on 29 March 2019. Retrieved from https://www.protacon.com/tuotekehitysorganisaatiot/rd-hankkeet-ja-simulointi/

Real-Time Target Machines. 2019. Speedgoat web-page. Accessed on 13 April 2019. Retrieved from https://www.speedgoat.com/products-services/real-time-target-machines

Robot Framework User Guide version 3.1.1. 2019. Robot Framwork web-page. Accessed on 10 April 2019. Retrieved from

http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.htm
l

Robot Framework. 2019. Robot Framework official web-pages. Accessed on 4 March 2019. Retrieved from https://robotframework.org/

Test Automation Frameworks. 2019. Smartbear. Accessed on 29 March 2019. Retrieved from https://smartbear.com/learn/automated-testing/test-automation-frameworks/

Автоматизация тестирования [Test Automation]. 2018. PerformanceLab. Accessed on 5 March 2019. Retrieved from https://www.performance-lab.ru/avtomatizacija-testirovanija

Автоматизированное тестирование [Automated Testing]. 2018. Wikipedia, the free encyclopedia. Accessed on 29 March 2019. Retrieved from https://ru.wikipedia.org/wiki/Автоматизированное_тестирование

Программно-аппаратное моделирование [Software and hardware modeling]. 2018. Wikipedia, the free encyclopedia. Accessed on 29 March 2019. Retrieved from https://ru.wikipedia.org/wiki/Программно-аппаратное_моделирование

Средства для разработки ПО [Software Development Tools]. Jira Software. 2019. Atlassian. Accessed on 7 March 2019. Retrieved from https://ru.atlassian.com/software/jira/features

Appendices

Appendix 1. getDataFromIOFile function

```
private void getDataFromIOFile()
{
    signalsFromIOFile.Clear();
    if (File.Exists(fileName))
    {
        string line;
        StreamReader sr = new StreamReader(fileName);
        while ((line = sr.ReadLine()) != null)
        {
            string[] data = line.Split('\t');
            signalsFromIOFile.Add(data[0], data[1].ToString());
        }
        sr.Close();
    }
    else
    {
        Console.WriteLine("File doesn't exists. Press button 'Set I/O signals and fill data'");
    }
    createInputOutputTables();
}
```

Appendix 2. Recording of data (programm code)

```
public partial class Form1 : Form
   {
     private void buttonRecordSignals_Click(object sender, EventArgs e)
        {
            GlobalVar.recording = true;
            StartRecording();
            buttonRecordSignals.Text = "Recording in progress";
            buttonRecordSignals.BackColor = Color.Red;
            buttonRecordSignals.Enabled = false;
            buttonStopRecord.Enabled = true;
            dataGridView2.Rows.Clear();
            dataGridView2.Refresh();
        }
        public static void StartRecording()
        {
            GlobalVar.listOfRecords.Clear();
            RecordThread thread = new RecordThread();
            Thread t = new Thread(new ThreadStart(thread.Recording));
            t.Start();
        }
}

public class RecordThread
    {
        public void Recording()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            while (GlobalVar.recording)
            {
                Dictionary<string, double> row = new Dictionary<string, double>();
                row.Add("Time", stopwatch.ElapsedMilliseconds);
                for (int i = 0; i < GlobalVar.readSignalBindings.Count; i++)
                {
                    TAPIReadSignal signal = (TAPIReadSignal)GlobalVar.readSignalBindings[i];
                    row.Add(signal.SignalID.ToString(), signal.ScaledValue);
                }
                GlobalVar.listOfRecords.Add(row);
                Thread.Sleep(GlobalVar.freq);
            }
            stopwatch.Stop();
        }
    }
```

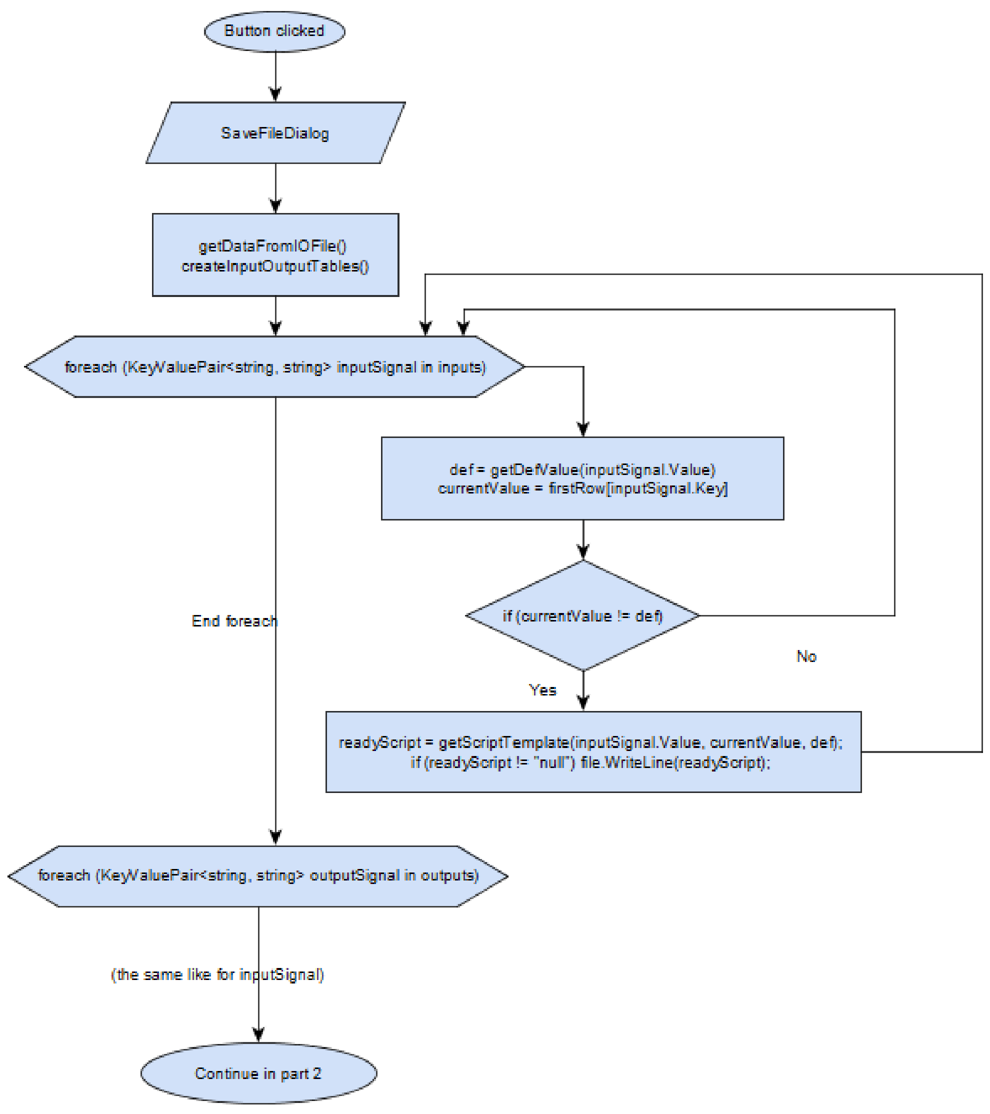
Appendix 3. Stop recording and display recorded data

```
private void buttonStopRecord_Click(object sender, EventArgs e)
{
    GlobalVar.recording = false;
    buttonRecordSignals.Text = "Start recording";
    buttonRecordSignals.BackColor = Color.Green;
    buttonRecordSignals.Enabled = true;
    buttonStopRecord.Enabled = false;
    ShowResults();
    labelData.Text = "Data from the last record";
}



public void ShowResults()
{
    dataGridView2.Rows.Clear();
    dataGridView2.Refresh();
    dataGridView2.ColumnCount = GlobalVar.readSignalBindings.Count + 1;
    GlobalVar.columns.Clear();
    int rowId = dataGridView2.Rows.Add();
    int col = 0;
    // Get all records line by line
    for (int i = 0; i < GlobalVar.listOfRecords.Count; i++)
    {
        List<string> list = new List<string>();
        // Every row in a table saved as dictionary.
        // Take one row and get key value pair for signals.
        Dictionary<string, double> signal = GlobalVar.listOfRecords[i];
        foreach (KeyValuePair<string, double> kvp in signal)
        {
            // In case of the first row dataGridView columns will be added
            if (i == 0)
            {
                dataGridView2.Columns[col].Name = kvp.Key;
                GlobalVar.columns.Add(kvp.Key);
                col++;
            }
            list.Add(kvp.Value.ToString());
        }
        string[] row = list.ToArray();
        dataGridView2.Rows.Add(row);
    }
}
```

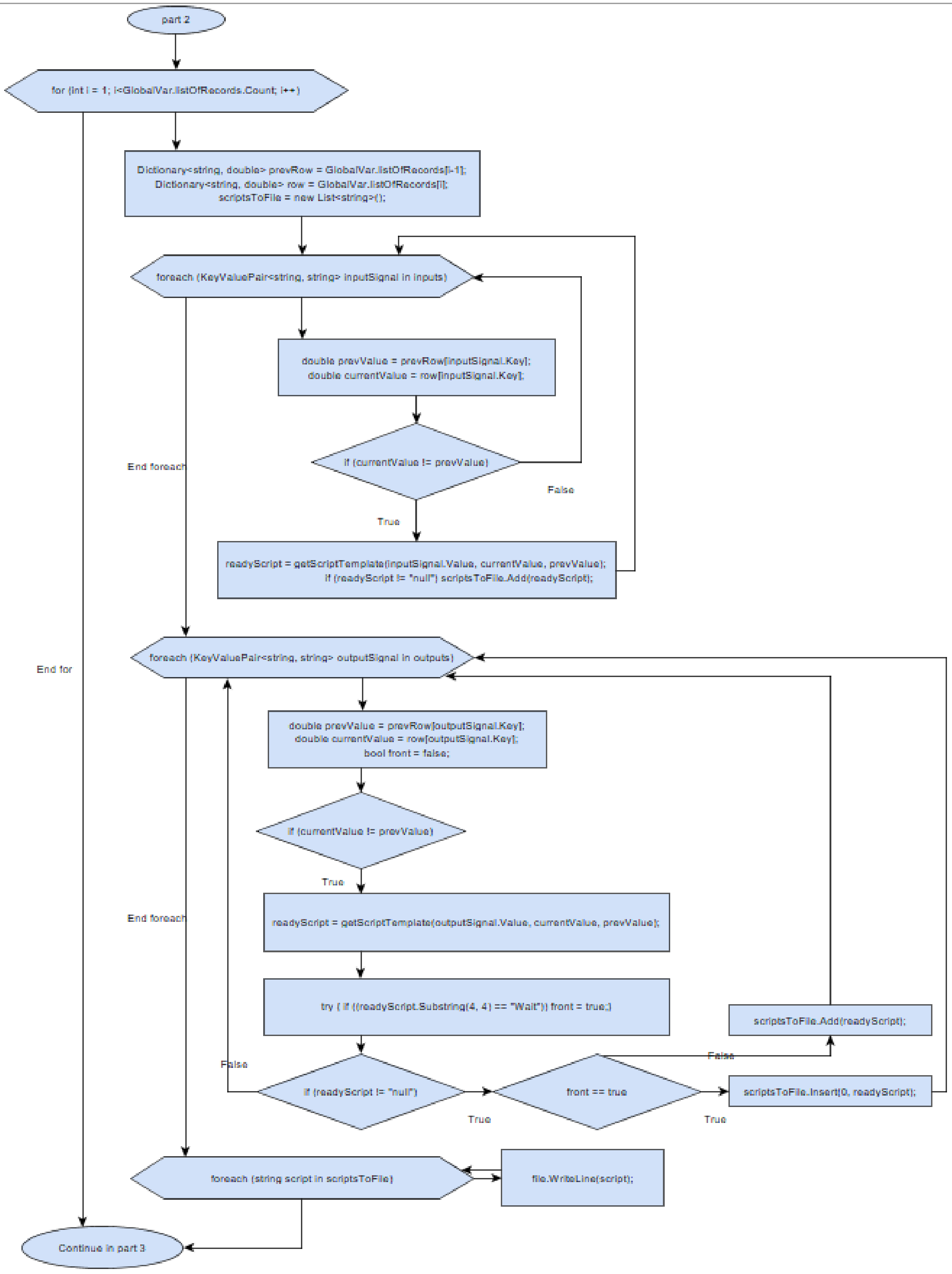
Appendix 4. Save recorded data to file

```
 private void buttonSave_Click(object sender, EventArgs e)
 {
     SaveFileDialog saveFileDialog1 = new SaveFileDialog();
     saveFileDialog1.Title = "Save Into File";
     saveFileDialog1.Filter = "Text file|*.txt";
     saveFileDialog1.ShowDialog();
     string fileName = saveFileDialog1.FileName;
     if (fileName != "")
     {
         // Meke the first row with columns names
         string colNames = "";
         foreach (string col in GlobalVar.columns)
         {
             if (colNames != "") colNames += "\t";
             colNames += col;
         }
         // Add columns names into the file
         var file = new StreamWriter(saveFileDialog1.FileName);
         file.WriteLine(colNames);
         // Add signal values into the file
         for (int i = 0; i < GlobalVar.listOfRecords.Count; i++)
         {
             string line = "";
             for (int j = 0; j < GlobalVar.columns.Count; j++)
             {
                 string key = GlobalVar.columns[j];
                 Dictionary<string, double> signal = GlobalVar.listOfRecords[i];
                 double value = signal[key];
                 if (line != "") line += "\t";
                 line += value;
             }
             file.WriteLine(line);
         }
         file.Close();
     }
}
```

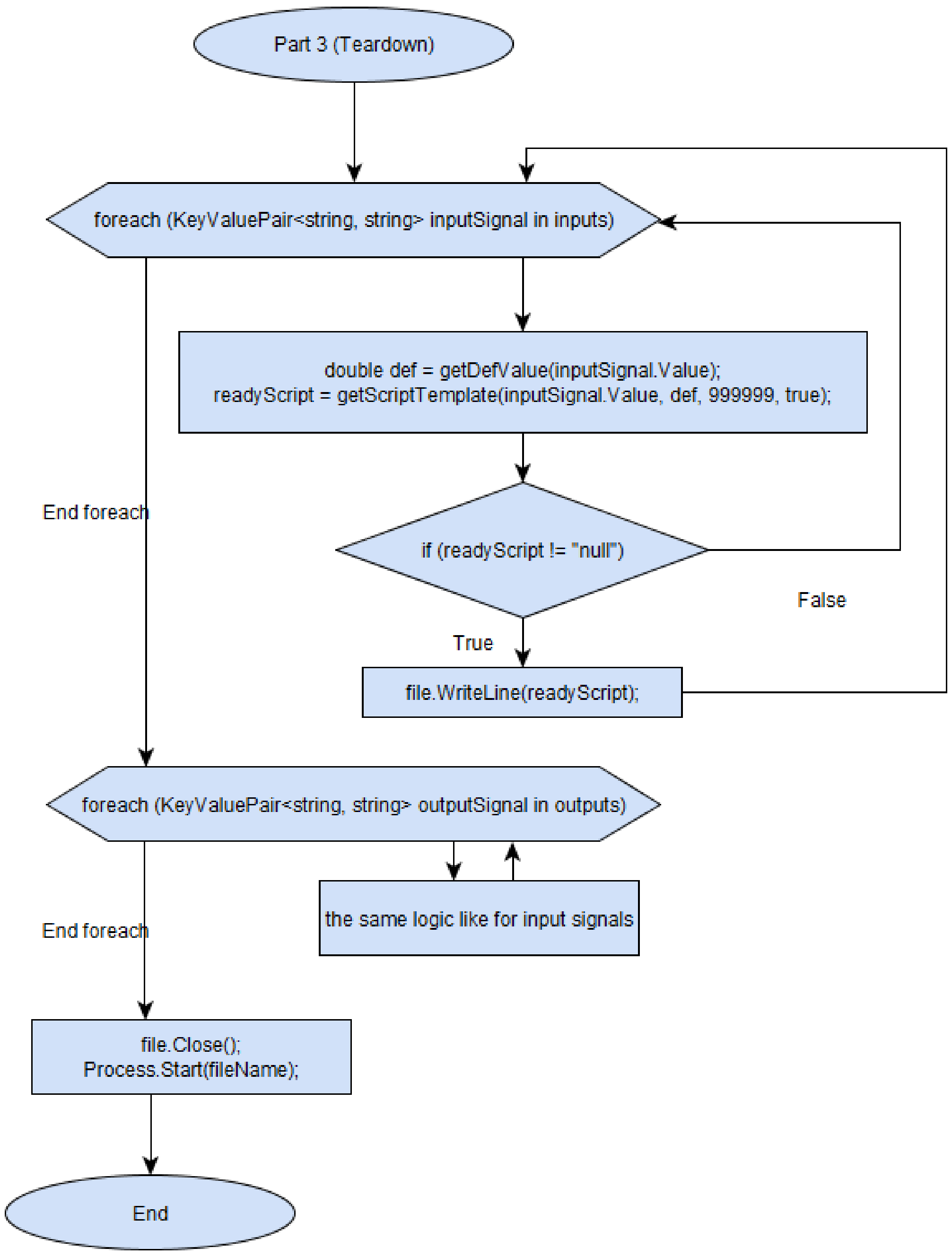
Appendix 5. Script creation diagram (part1)


Button clicked
SaveFileDialog
getDataFromIOFile()
createInputOutputTables()
foreach (KeyValuePair<string, string> inputSignal in inputs)
def = getDefValue(inputSignal.Value)
currentValue = firstRow[inputSignal.Key]
if (currentValue != def)
No
Yes
readyScript = getScriptTemplate(inputSignal.Value, currentValue, def);
if (readyScript != "null") file.WriteLine(readyScript);
End foreach
foreach (KeyValuePair<string, string> outputSignal in outputs)
(the same like for inputSignal)
Continue in part 2

Appendix 6. Script creation diagram (part 2)


part 2
for (int i = 1; i<GlobalVar.listOfRecords.Count; i++)
Dictionary<string, double> prevRow = GlobalVar.listOfRecords[i-1];
Dictionary<string, double> row = GlobalVar.listOfRecords[i];
scriptsToFile = new List<string>();
foreach (KeyValuePair<string, string> inputSignal in inputs)
double prevValue = prevRow[inputSignal.Key];
double currentValue = row[inputSignal.Key];
if (currentValue != prevValue)
End foreach
False
True
readyScript = getScriptTemplate(inputSignal.Value, currentValue, prevValue);
if (readyScript != "null") scriptsToFile.Add(readyScript);
End for
foreach (KeyValuePair<string, string> outputSignal in outputs)
double prevValue = prevRow[outputSignal.Key];
double currentValue = row[outputSignal.Key];
bool front = false;
if (currentValue != prevValue)
True
End foreach
readyScript = getScriptTemplate(outputSignal.Value, currentValue, prevValue);
try { if ((readyScript.Substring(4, 4) == "Wait")) front = true;}
scriptsToFile.Add(readyScript);
False
False
if (readyScript != "null")
front == true
scriptsToFile.Insert(0, readyScript);
True
True
foreach (string script in scriptsToFile)
file.WriteLine(script);
Continue in part 3

Appendix 7. Script Creation diagram (part 3)


Part 3 (Teardown)
foreach (KeyValuePair<string, string> inputSignal in inputs)
double def = getDefValue(inputSignal.Value);
readyScript = getScriptTemplate(inputSignal.Value, def, 999999, true);
if (readyScript != "null")
False
True
file.WriteLine(readyScript);
End foreach
foreach (KeyValuePair<string, string> outputSignal in outputs)
the same logic like for input signals
End foreach
file.Close();
Process.Start(fileName);
End

Appendix 8. Additional functions (programm code)

```
private void getDataFromIOFile()
{
    signalsFromIOFile.Clear();
    if (File.Exists(fileName))
    {
        string line;
        StreamReader sr = new StreamReader(fileName);
        while ((line = sr.ReadLine()) != null)
        {
            string[] data = line.Split('\t');
            signalsFromIOFile.Add(data[0], data[1].ToString());
        }
        sr.Close();
    }
    else
    {
        Console.WriteLine("File doesn't exists. Press button 'Set I/O signals and fill data'");
    }
    createInputOutputTables();
}



public void createInputOutputTables()
{
    inputs.Clear();
    outputs.Clear();
    foreach (string colName in GlobalVar.columns)
    {
        try
        {
            string value = signalsFromIOFile[colName];
            if (value.Substring(0, 1) == "1")
                inputs[colName] = value.Substring(3);
            if (value.Substring(0, 1) == "0")
                outputs[colName] = value.Substring(3);
        }
        catch
        {
            Console.WriteLine(colName + " not found");
        }

    }
}
```

Appendix 9. getScriptTemplate function

```
private string getScriptTemplate(string data, double currentValue, double prevValue)
        {
            string value = "";
            var items = data.Replace("    ", "*");
            string[] array = items.Split('*');
            string returnText = "empty";
            bool isCase = false;
            foreach (string item in array)
            {
                if (item.Substring(0,4) == "case")
                {
                    isCase = true;
                    if (item.Substring(4,1) == "=")
                    {
                        int symbol = item.IndexOf(':');
                        string val = item.Substring(5, symbol-5);
                        if (Double.Parse(val) == currentValue)
                        {
                            value = item.Substring(symbol+1);
                            returnText = "     " + array[0] + " " + value;
                        }
                    }
                    else if (item.Substring(4, 1) == ">")
                    {
                        int symbol = item.IndexOf(':');
                        string val = item.Substring(5, symbol - 5);
                        if (currentValue > Double.Parse(val) && prevValue <= Double.Parse(val) && returnText
== "empty")
                        {
                            value = item.Substring(symbol + 1);
                            returnText = "     " + array[0] + " " + value;
                        }
                    }
                    else if (item.Substring(4, 1) == "<")
                    {
                        int symbol = item.IndexOf(':');
                        string val = item.Substring(5, symbol - 5);
                        if (currentValue < Double.Parse(val) && prevValue >= Double.Parse(val) && returnText
== "empty")
                        {
                            Console.WriteLine("Script making: Prev: {0}, current: {1}, script: {2}",
prevValue, currentValue, item);
                            value = item.Substring(symbol + 1);
                            returnText = "     " + array[0] + " " + value;
                        }
                    }
                }
            }
            if (isCase && returnText == "empty") returnText = "null";
            else returnText = "     " + array[0] + " " + value;
            return returnText;
        }
```

Appendix 10. Manual test VS. generated script

Tractor manual test description:	Generated robot test script
	Test it
Set Shuttle Lever reverse position	Set Shuttle Lever neutral
	Parking Brake Status Should Be off
	Parking Brake Indicator State Should Be off
	Set Shuttle Lever reverse
Wait Until Speed is above 5km/h	Wait Until Value Is Above CANSpeed 5
Press Clatch Pedal 100%	Set Acceleration Pedal 0
	Set Clutch Pedal 100
Press Brake pedal 50%	Press Brake 50
	Wait Until Value Is Below CANSpeed 1
Set Shuttle Lever forward position	Set Shuttle Lever neutral
	Set Shuttle Lever forward
Release Clatch Pedal	Set Clutch Pedal 0
Release Brake pedal	Press Brake 0
Press Acceleration pedal 50%	Set Acceleration Pedal 50
Wait Until Speed is above 10km/h	Wait Until Value Is Above CANSpeed 5
	Wait Until Value Is Above CANSpeed 10
Release Acceleration pedal	Set Acceleration Pedal 0
Set Shuttle Lever park position	Set Shuttle Lever neutral
	Set Shuttle Lever park
Check what parking brake indicator is on	Parking Brake Status Should Be on
	Parking Brake Indicator State Should Be on
	Teardown
	Set Acceleration Pedal 0
	Set Clutch Pedal 0
	Set Shuttle Lever park
	Press Brake 0
	Parking Brake Status Should Be on
	Parking Brake Indicator State Should Be on

Appendix 11. Generated file

```
*** Settings ***
Library            Remote    http://${ADDRESS}:${PORT}
Documentation      This test was generated automatically on the base
of manual test
Test Teardown      Teardown


*** Variables ***
${ADDRESS}         %{ROBOT_SERVER_ADDRESS}
${PORT}            8265

*** Test Cases ***
Test it
    Set Shuttle Lever  neutral
    Parking Brake Status Should Be  off
    Parking Brake Indicator State Should Be  off
    Set Shuttle Lever  reverse
    Wait Until Value Is Above  CANSpeed  5
    Set Acceleration Pedal  0
    Set Clutch Pedal  100
    Press Brake  50
    Wait Until Value Is Below  CANSpeed  1
    Set Shuttle Lever  neutral
    Set Shuttle Lever  forward
    Set Clutch Pedal  0
    Press Brake  0
    Set Acceleration Pedal  50
    Wait Until Value Is Above  CANSpeed  5
    Wait Until Value Is Above  CANSpeed  10
    Set Acceleration Pedal  0
    Set Shuttle Lever  neutral
    Set Shuttle Lever  park
    Parking Brake Status Should Be  on
    Parking Brake Indicator State Should Be  on

***Keywords ***
Teardown
    Set Acceleration Pedal  0
    Set Clutch Pedal  0
    Set Shuttle Lever  park
    Press Brake  0
    Parking Brake Status Should Be  on
    Parking Brake Indicator State Should Be  on
```