



jamk.fi

Performance Testing Smart Metering Systems

Tools, Automated Tests and Reporting

Joel Kortelainen

Bachelor's thesis

May 2017

Technology, communication and transport

Degree Programme in Software Engineering

Author(s) Kortelainen, Joel	Type of publication Bachelor's thesis	Date May 2017
	Number of pages 38	Language of publication: English
		Permission for web publication: x
Title of publication Performance Testing Smart Metering Systems Tools, Automated Tests and Reporting		
Degree programme Software Engineering		
Supervisor(s) Esa Salmikangas, Jouni Huotari		
Assigned by Landis+Gyr, Jyskä		
<p>Description</p> <p>Landis+Gyr's R&D department in Jyskä had a need for high-level, automated performance tests for the Gridstream AIM smart metering system. Landis+Gyr assigned a task to develop a set of automated tests and a way to store and visualize the gathered data for history and regression purposes.</p> <p>The objective was to design and develop a set of high-level, automated performance tests for specific parts of the Gridstream AIM system according to the requirement specification as well as a system for storing and displaying the gathered performance data. The tests were developed using Robot Framework, MongoDB was used as a data storage and a web server written in Node.js served as a back-end. Data visualization was done using C3.js chart library and the data was retrieved from the database by using a simple REST API. The tests were executed periodically as a scheduled Jenkins jobs.</p> <p>As a result, the required test cases were developed according to requirement specification. Jenkins executes the Robot Framework test suites automatically every night, the desired data is gathered and stored to MongoDB. The overall test execution status is automatically reported to TestRail as a test teardown procedure. Node.js-based web server serves static web sites with C3.js graph templates that receive the gathered performance data from MongoDB via REST API.</p> <p>Robot Framework is an excellent tool for developing high-level test automation. Due to a variety of different testing libraries it is possible to develop test automation regardless of the programming language the system under test is based on. Building an architecture that handles test execution, data storing and visualization automatically, can easily be done by using only open-source tools and technologies. By constant performance monitoring, the chances of uncovering new software bugs and regressions are greatly increased, which reduces the need for repetitive manual labor.</p>		
Keywords (subjects)		
Test automation, Performance testing, Robot Framework, Jenkins, Windows		
Miscellaneous		

Tekijä(t) Kortelainen, Joel	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Toukokuu 2017
	Sivumäärä 38	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty: x
Työn nimi Performance Testing Smart Metering Systems Tools, Automated Tests and Reporting		
Tutkinto-ohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) Esa Salmikangas, Jouni Huotari		
Toimeksiantaja(t) Landis+Gyr, Jyskä		
Tiivistelmä <p>Landis+Gyr Oy:n Jyskän tuotekehitysosastolla oli tarve korkean tason automatisoiduille suorituskykytesteille Gridstream AIM -älymittausjärjestelmälle. Landis+Gyr Oy antoi tehtäväksi kehittää joukon automaattisia testejä sekä tavan tallentaa ja visualisoida kerätty data historia- ja regressiotarkoituksiin.</p> <p>Tavoitteena oli suunnitella ja kehittää joukko korkean tason automatisoituja suorituskykytestejä Gridstream AIM -järjestelmän graafisen käyttöliittymän tiettyihin toiminnallisuuksiin vaatimusmäärittelyn mukaisesti sekä kehittää järjestelmä kerätyn tiedon tallentamiseen sekä esittämiseen. Testit kehitettiin käyttämällä Robot Framework -ohjelmistokehystä, MongoDB:tä käytettiin tietovarastona ja Node.js:llä kirjoitettu web-palvelin toimi palvelun taustana. Tietojen visualisointi suoritettiin käyttämällä C3.js -kaaviokirjastoa ja tiedot haettiin tietokannasta käyttämällä yksinkertaista REST-rajapintaa. Testejä suoritettiin säännöllisesti ajastettuina Jenkins-töinä.</p> <p>Tämän tuloksena vaaditut testitapaukset kehitettiin vaatimusmäärittelyn mukaisesti. Jenkins -automaatiopalvelin suorittaa Robot Framework -testit joka yö määrätyllä aikavälillä, halutut tiedot kerätään ja tallennetaan MongoDB:hen. Testien suoritustila raportoidaan automaattisesti TestRail-palveluun testin purkumenetelmänä. Node.js -pohjainen web-palvelin tarjoilee staattisia web-sivuja, joissa on C3.js -kaaviomallipohjat, jotka vastaanottavat kerättyä tietoa Mongo-DB:stä REST-rajapinnan avulla.</p> <p>Robot Framework on erinomainen työkalu korkean tason testiautomaation kehittämiseen. Erilaisten testikirjastojen avulla on mahdollista kehittää testiautomaatiota riippumatta testattavan järjestelmän ohjelmointikielestä. Arkkitehtuurin, joka suorittaa testit, tallentaa ja visualisoi tiedot automaattisesti, voidaan helposti rakentaa käyttämällä vain avoimen lähdekoodin työkaluja ja tekniikoita. Suorituskyvyn jatkuvalla seuraamisella uusien ohjelmistovirheiden ja regressioiden löytäminen helpoituu, mikä puolestaan vähentää toistuvan, manuaalisen työn tarvetta.</p>		
Avainsanat (asiasanat) Testiautomaatio, Suorituskykytestaus, Robot Framework, Jenkins, Windows		
Muut tiedot		

Contents

1	Introduction.....	8
2	Basis of the thesis.....	9
2.1	Theoretical background.....	9
2.1.1	Test automation	9
2.1.2	Performance testing.....	9
2.2	Landis+Gyr	10
2.3	Gridstream AIM	10
2.4	Thesis background and assignment.....	11
2.5	Objectives of the thesis	12
2.5.1	Concrete objectives.....	12
2.5.2	Other objectives	12
3	Tools and their role in development.....	13
3.1	Git	13
3.2	Gerrit.....	14
3.3	JIRA	15
3.4	Robot Framework.....	16
3.4.1	Overview	16
3.4.2	Test data tables	16
3.4.3	Best practices	16
3.5	Jenkins	18
3.6	TestRail	19
3.7	MongoDB.....	20
3.8	Node.js.....	21
3.9	C3.js.....	23

4	Workflow	24
4.1	Developing automated tests	24
4.1.1	Research	24
4.1.2	Development	25
4.1.3	Review	28
4.2	Data gathering and storing	28
4.3	Displaying the results	29
4.4	Automated test execution	30
4.4.1	Jenkins plugins.....	30
4.4.2	Slave machine setup.....	31
4.4.3	Creating jobs.....	32
5	Evaluation.....	34
6	Further development	35
7	Conclusion	36
	References.....	37

Figures

Figure 1. Mainline Branch Strategy	13
Figure 2. Gerrit Workflow Model	14
Figure 3. Robot Framework - Appropriate abstraction level	17
Figure 4. Robot Framework - Too low abstraction level.....	17
Figure 5. TestRail overview	20
Figure 6. MongoDB example document	21
Figure 7. A simple Node.js web server	22
Figure 8. Combination chart sample	23
Figure 9. Test case breakdown example	24
Figure 10. Folder structure.....	25
Figure 11. Argument file example	26
Figure 12. Batch script example	27
Figure 13. Jenkins Robot Framework Plugin	30
Figure 14. Jenkins Build Monitor Plugin	31
Figure 15. Jenkins Slave Launch Options	32
Figure 16. Job scheduling example	33
Figure 17. Project's overall architecture	34

Tables

Table 1. Agile delivery vehicles	15
Table 2. Robot Framework test data tables.....	16

Terms

AMM

Advanced Metering Management

AngularJS

An open-source front-end web application framework.

API

Application Programming Interface is a set of tools, protocols and subroutine definitions for building an application or software system.

Gerrit

A web-based team code collaboration tool closely integrated with Git (see Git) that enables developers to review, approve or reject each other's modifications on their source code.

Git

A distributed version control system (see VCS) that allows developers to work on a specific project without them sharing a common network.

C3.js

A D3.js-based (see D3.js) reusable chart library.

CD

Continuous Delivery is an approach where software is produced in short cycles to ensure a reliable release at any given time.

CI

Continuous Integration is a practice of testing and merging all isolated code changes to a shared mainline several times a day.

Cron

Linux utility which schedules a command or script on a server to run automatically at a specified time and date.

CSS

Cascading Style Sheets is a style sheet language most often used to set the visual style of web pages.

D3.js

Data-Driven Documents is a JavaScript (see JavaScript) library for manipulating data-based documents while providing powerful data visualization tools.

EMEA

Europe, the Middle East and Africa

Express.js

An open-source web application framework for Node.js (see Node.js).

HES

Head End System

IDE

An integrated development environment is a software application usually consisting of a source code editor, build automation tools and a debugger.

I/O

Input/output is the communication between an information processing system and the outside world.

Java

A class-based and object-oriented general-purpose programming language.

JavaScript

Dynamic, untyped and interpreted programming language used by web browsers to display dynamic content.

JIRA

Issue tracking and project management product , developed by Atlassian.

Jenkins

An open-source automation server written in Java (see Java).

JSON

JavaScript Object Notation is an open-standard format for transmitting data objects consisting of attribute-value pairs. JSON is mostly used for asynchronous browser/server communication.

MEAN

A free and open-source software stack consisting of MongoDB (see MongoDB), Express.js (see Express.js), AngularJS (see AngularJS) and Node.js (see Node.js).

MongoDB

An open-source cross-platform document-oriented database (see Database) program that uses JSON-like (see JSON) documents.

Node.js

Open-source JavaScript (see JavaScript) runtime environment for developing server tools and applications.

NoSQL

"Not only SQL" (see SQL) database provides a mechanism to manage data which is modeled differently from the tabular relations used in relational databases.

npm

The default package manager for Node.js (see Node.js).

Python

A general-purpose, high-level programming language.

QA

Quality assurance

R&D

Research and development.

Software framework

An abstraction that enables application-specific software by providing a software with generic functionality which can be selectively changed by additional code.

SQL

Structured Query Language is a domain-specific language designed for managing data stored in a relational database management system.

Test case

A set of conditions used to determine if a feature in an application or software system is working as it was originally intended.

TestRail

A web-based test case management software for managing, tracking and organizing testing efforts.

Test suite

A collection of test cases (see Test case) used to show that an application or software system has some specified set of behaviours.

VCS

Version Control System tracks and provides control over changes to source code and enables developers to quickly switch between different versions of their software

1 Introduction

Testing has always been an integral part of software development. Testing helps to ensure that the product in development meets the requirements of its design such as responding correctly to given inputs, performing its functions within an acceptable time and that the overall usability is sufficient. However, testing large entities manually can be extremely taxing in terms of time, money and resources. That is where automation comes in. When automation handles the boring and repetitive testing tasks, the human resources are freed to analyze the results and solve the possible problems instead of consuming the precious development time trying to locate said problems.

An example of these large entities is the Landis+Gyr's smart metering solution. Electricity meters are found in millions of homes and their proper functioning is crucial for both, the client and the supplier. The electricity companies around the world use smart metering systems to remotely read data from thousands upon thousands of meters simultaneously and there is little to no room for errors. The data that is collected should be valid and the operation itself should take as little time as possible. One way to ensure the good quality of the product is comprehensive automated testing for regression, acceptance and performance purposes.

This thesis introduces the basic workflow of developing high-level performance test automation for smart metering systems, the tools and technologies that are being used in development as well as a solution for automatically reporting the test results and the gathered data. It is worth noting that due to a non-disclosure agreement made with Landis+Gyr, this thesis does not go into detail on the functions, business processes or design decisions behind Landis+Gyr's smart metering systems, but describes them only superficially.

2 Basis of the thesis

2.1 Theoretical background

2.1.1 Test automation

In software testing, test automation is a process that utilizes a special software to execute pre-scripted tests on a software application and compares the actual results of the tests with the predicted ones. Test automation is ideal for repetitive but necessary tasks and it can drastically decrease test execution time compared to manual testing. Test execution speed is not the only benefit test automation can offer compared to manual testing, as it also negates the possibility of human error by always executing the tests in an identical manner.

Test automation provides an efficient way for software development and quality assurance teams to catch possible defects of their products, as the tests can be executed off-hours and the results examined in the morning.

Test automation has a range of uses such as functional acceptance testing, regression testing and integration testing, however, there are some cases where manual testing excels such as usability testing and exploratory testing.

2.1.2 Performance testing

In software testing, performance testing is a type of non-functional testing that is used to determine how fast some aspect of a system performs under a particular workload. (What is Performance testing in software, 2017). Performance testing can be used to measure parameters such as throughput, bandwidth, data transfer rate or efficiency.

The scope of this thesis focuses mainly on high-level performance testing. This means measuring the execution time of certain user interface operations, data transfer rates and system's overall reliability.

2.2 Landis+Gyr

Landis+Gyr has been in the energy business for more than a century. Their meters and solutions empower utilities and end-customers around the world to improve their energy efficiency, reduce their energy costs and contribute to a sustainable use of resources. Landis+Gyr is the largest global player in smart metering with one of the broadest portfolios in the industry. (Landis+Gyr, 2017)

Landis+Gyr has 45 companies in over 30 countries and it is headquartered in Zug, Switzerland. Since 2011, Landis+Gyr has been an independent growth platform of the Japanese Toshiba Corporation. (Landis+Gyr, 2017)

Founded in 1999, Landis+Gyr's software R&D site in Jyskä, Finland is one of the global technology centers that focuses on remote reading systems and smart metering solutions. The Jyskä site currently employs over 200 IT professionals, most of whom work in development.

2.3 Gridstream AIM

Gridstream AIM is a part of Landis+Gyr's Gridstream smart metering solution that provides data management capabilities, a task flow engine and data validation beyond basic HES functions. With a single, integrated software the utility can manage its AMM infrastructure, support its network management operations and enhance its business processes. (Landis+Gyr – Gridstream AIM, 2017)

Support for utility processes in Gridstream AIM. (Landis+Gyr – Gridstream AIM, 2017)

- Business processes – billing, balance settlement, customer service, contract management and new business development.
- Network management – network monitoring, load management and network investment planning.
- Smart metering operations – automated data collection and management, device asset management and troubleshooting.

Gridstream AIM is based on openness and modularity. It is designed for the ever-changing energy market to help companies collect and manage their metering data efficiently, process it in a flexible manner and transfer information between various parties effortlessly. It offers a single source for metering data that can be transferred to other systems. Today, Gridstream AIM runs over 2 million metering points in EMEA. (Landis+Gyr – Gridstream AIM, 2017)

2.4 Thesis background and assignment

In the summer of 2016, the author spent four months as an intern for the Solution Integration Team at Landis+Gyr's Jyskä site developing test automation for the Gridstream AIM system. In that four-month period a fairly good understanding of how different parts of the system function as well as an immense amount of knowledge about test automation in general was gained. There was not have enough time to complete the full five-month internship period; therefore, as the summer was ending there was a discussion with the manager about the possibility of coming back to finish the internship period and writing the thesis there. An informal agreement was made on returning back to the company after New Year. The remainder of the internship was started in January 2017 and after three weeks, the internship was over and it was time to start thinking about the subject of the Bachelor's thesis. There had been some talks about performance testing, however, nothing had been formally agreed upon.

The AIM development team had a need for a set of high-level performance tests for the Gridstream AIM system that would gather performance data for history, acceptance and regression purposes. The tests should be automated and they should be executed at regular intervals. Another requirement was that the data the tests gathered should be stored and displayed in a way that it is easy to access and analyze. As the author was already somewhat familiar with the system and with test automation, that assignment seemed like a perfect subject for the thesis.

2.5 Objectives of the thesis

2.5.1 Concrete objectives

The main objective of the thesis was to develop a set of high-level automated performance test cases for Landis+Gyr's Gridstream AIM smart metering system. The tests were to cover the performance of different parts of the data transfer chain, like installing a new meter to the system, remotely reading data from the meter, validating the data and remotely connecting and disconnecting the meter. The results of these tests are stored and displayed in order to see how the performance of the system develops over time.

Everything mentioned above, is to be fully automated, which means that once the testing environment has been built and configured, everything from test execution, data gathering and storing, to reporting the results and displaying them will take place automatically at regular intervals.

2.5.2 Other objectives

The author's personal objectives for this thesis were not only to learn more about test automation, different development tools and smart metering systems in general, but also to learn to create tools to help with test automation development. Test automation is not a new phenomenon in the information technology business by any means and a staggering number of different open-source tools and libraries can be found from the internet. But every now and then a situation comes along that requires building one's own tools in order to proceed with the development. Usually this means creating a new library or altering an existing one. That said, the main goal, personally speaking, was to build the confidence and skills so as to be able to create an own solution to a problem if an existing one is nowhere to be found.

3 Tools and their role in development

3.1 Git

Git and other version control systems are crucial for any software project with multiple team members as they offer an easy way to track modifications made on the source code and fast switching between different versions, or branches, of the software. Git offers a variety of different branching strategies for different situations, however, the simplest, yet the most effective strategy for small to medium sized teams is the mainline branch strategy. The developers constantly commit their work into a single, central branch, also called the master branch. The master branch should only contain fully tested and quality work and should never be broken (Git Branching Strategies, 2015).

By having their own copy of the master branch, each developer has the freedom to experiment and test new features without the fear of breaking existing implementations. When a developer decides that their work is done and that everything works as intended, they commit their changes, and usually after the changes are followed by a review and acceptance (Chapter 3.2 Gerrit), after which their work gets merged into the master branch (Figure 1).

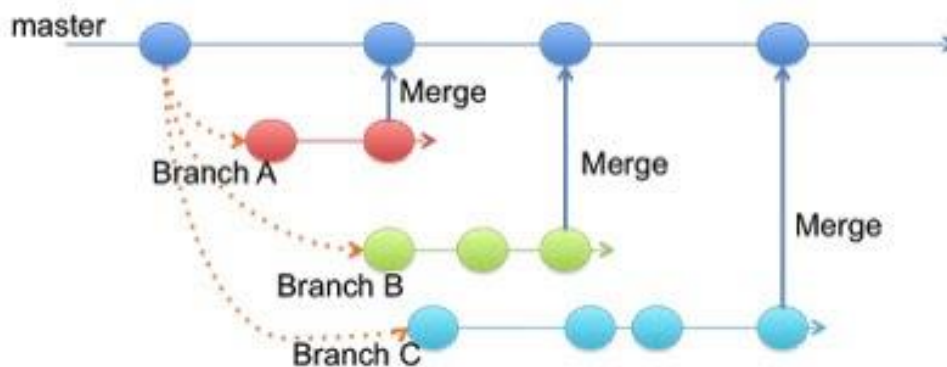


Figure 1. Mainline Branch Strategy (Git Branching Strategies, 2015)

3.2 Gerrit

Gerrit is a web-based code collaboration tool that is heavily integrated with Git. It supports comparing old and new versions of code with syntax highlighting and colored differences as well as posting comments to specific parts of the code for others to see (Gerrit Code Review, 2017). The basic Gerrit workflow goes as follows: when a developer commits and pushes their changes to the Gerrit server, the changes are automatically put into a temporary review branch where the workflow engine enforces the rules before the changes can be merged into a permanent repository. These rules usually involve automatic building and testing, checks to see if the changes conflict with other commits and also human code reviews (Figure 2).

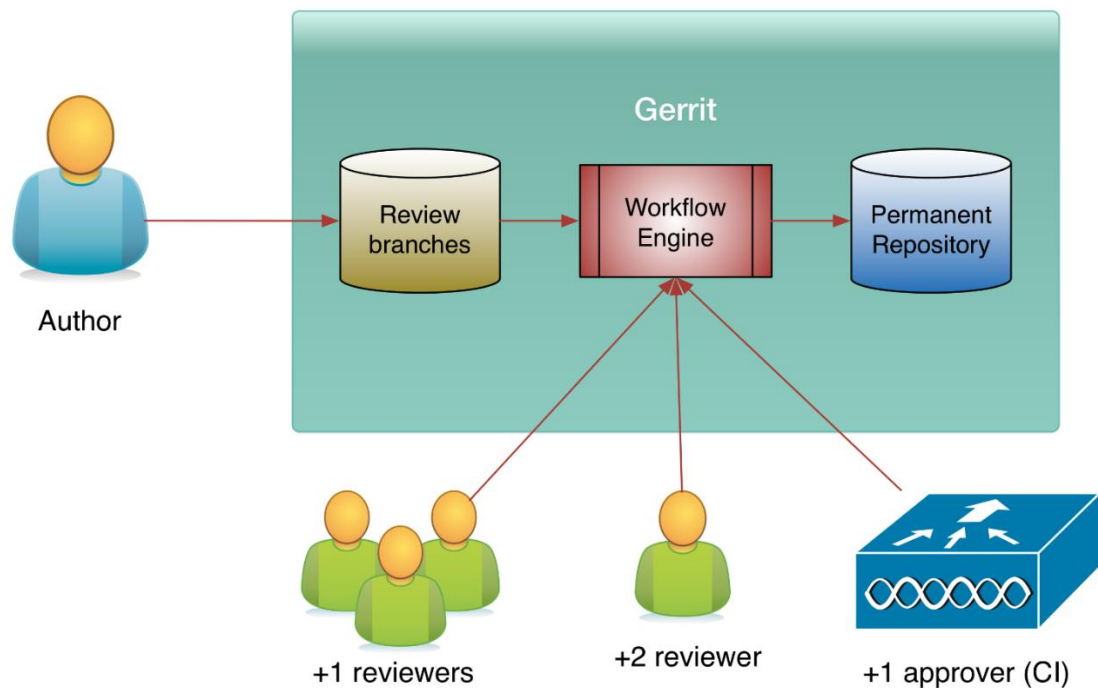


Figure 2. Gerrit Workflow Model (Gerrit Workflow, 2014)

Gerrit has review levels ranging from -2 to +2. The -1 and +1 level are opinions, e.g. "looks good to me, but someone else must approve it" and the -2 and +2 level are for blocking or approving the changes. In order for a change to be accepted, it must have at least one +2 vote and no -2 votes. If these conditions are met, a submit button will become available, enabling the changes to be merged with a permanent repository. (Gerrit Code Review – A Quick Introduction, 2017)

3.3 JIRA

According to Atlassian, the developer of JIRA, JIRA is used for issue tracking and project management by over 60,000 companies in 122 countries across the globe. (Atlassian – Customers, 2017). JIRA supports SCRUM and any other agile methodologies by offering agile boards and reports as well as tools for planning, tracking and managing agile software development projects within a single tool. (Atlassian – Agile, 2017).

Agile development uses four so-called delivery vehicles in order to maintain the structure in a project: epics, stories, versions and sprints (Table 1).

Table 1. Agile delivery vehicles (Atlassian - Delivery Vehicles, 2017)

Epic	Story	Version	Sprint
Large body of work, contains stories	Smallest unit of work, also known as a task	The release of software to the customer	Iteration where team does the work

In large projects with multiple team members, keeping track on what everyone is working on and at the same time maintaining a clear picture of the overall progress can be a gruesome task without relevant tools. However, by breaking the work down to smaller tasks and giving them story points which describe the amount of work needed to complete the given task (i.e. one story point equals one day of work), managing the project becomes considerably easier as it now can be calculated what tasks fit into a sprint and what tasks do not. For example, there is a project with a backlog full of tasks, five team members working for eight hours a day, five days a week and a need to plan a sprint lasting for two weeks. Thus, there is a total of 50 working days at the team's disposal, and if one working day equals one story point, the maximum of 50 story points of work can be fit in one sprint. Additionally, given that the backlog is full of tasks, the team can start picking tasks to add to the sprint in priority order until there is a sprint that is both doable and contains enough work for everybody. In conclusion, JIRA and other project management tools are not just for managers, but they also help individual developers to stay on track on what they are doing and how their project is progressing.

3.4 Robot Framework

3.4.1 Overview

Robot Framework is a generic test automation framework for acceptance and acceptance-driven test development. It has easy-to-use tabular test data syntax and it utilizes the keyword-driven testing approach. Its testing capabilities can be further extended with bundled and self-made test libraries implemented either with Python or Java. (Robot Framework, 2017).

3.4.2 Test data tables

Robot framework's test data is structured in four types of tables (Table 2). In small test suites, all four tables are usually defined in a single file, in larger suites however, each table is usually defined in a separate file in order to reduce clutter and make the tests more readable and easier to maintain.

Table 2. Robot Framework test data tables (Robot Framework User Guide, 2017)

Table	Used for
Settings	Importing test libraries, resource files and variable files. Defining metadata for test suites and test cases.
Variables	Defining variables that can be used elsewhere in the test data.
Test cases	Creating test cases from available keywords
Keywords	Create user keywords from existing lower-level keywords

3.4.3 Best practices

The following section demonstrates some of the best practices in Robot Framework test development. These practices are based on the author's own experiences as well as Pekka Klärck's, the lead developer of Robot Framework, general guidelines on how to develop a good test (Robot Framework Dos and Don'ts, 2014).

Naming

Like in traditional programming, good naming plays a very important role as it makes the test cases easier to understand and maintain. Be it test suites, test cases, keywords, resource files or variables, naming should be consistent and explicit. A good general rule for naming is that names should tell "what", not "how".

Appropriate abstraction levels

Abstraction levels can make or break the readability of a test suite. The appropriate abstraction level is in the eye of the developer and it can be, at times, hard to determine. Below (Figures 3 & 4) are examples of appropriate and too low abstraction levels, and the differences in readability are fairly clear.

```

*** Test Cases ***
Logging In To Facebook Should Succeed
  Open Browser To Facebook Login Page
  Input Username    test_user
  Input Password   test_password
  Submit Credentials
  Welcome Page Should Be Open
  [Teardown]      Close Browser

```

Figure 3. Robot Framework - Appropriate abstraction level

```

*** Test Cases ***
Logging In To Facebook Should Succeed
  Open Browser    ${URL}    ${BROWSER}
  Maximize Browser Window
  Title Should Be    Facebook login
  Input Text    username_field    test_user
  Input Text    password_field    test_password
  Click Button    login_button
  Title Should Be    Facebook
  Location Should Be    ${FACEBOOK_MAIN_URL}
  [Teardown]      Close Browser

```

Figure 4. Robot Framework - Too low abstraction level

Test setup and teardown

In short, a test setup is a keyword that is executed before a test case, and a test teardown is executed after a test case. The more important of the two is teardown, as it is executed even if the test case fails allowing the use of proper clean-up activities. Setups and teardowns can be defined at suite level, test level, or even at keyword level.

Move complex logic to libraries

Complex logic within a test case reduces readability and makes the test case harder to maintain. Therefore, logic should be moved to a library when possible, as this hides the complexity from the test case and the same result can be achieved by using a single keyword. However, the logic within a library should be made as generic as possible so that when a need for similar functionality arises, the same logic can be used again, without any library modifications.

Avoid dependencies between test cases

Even though "chaining" test cases (successful execution of a test case requires that previous test case has also succeeded) might seem tempting at first, however, it is generally considered as a bad practice. If the test cases have dependencies to one another, the whole test suite needs to be executed when the results of specific test cases are wanted. It should be possible to execute every test case within a test suite separately without any prerequisite test cases.

Synchronize with polling

Robot Framework has multiple ways of synchronizing test execution, the most notable being Sleep and Wait Until X -keywords. Sleep pauses the test execution until a specified amount of time has passed, whereas Wait Until X checks for a desired result at regular intervals. Using Sleep is generally considered as a bad practice, as it forces the test to pause even if there are no impediments present that would hinder the execution. The most used Wait Until X -keyword, Wait Until Keyword Succeeds, is a much more flexible approach to synchronizing the test execution as it allows defining the maximum wait time, the desired polling rate and the keyword that needs to succeed before continuing with the execution.

3.5 Jenkins

Jenkins is an open-source automation server that is mainly used as a continuous integration server or as a continuous delivery hub. It offers a wide range of plugins that help to integrate Jenkins with most of the tools used in CI/CD toolchain. (Jenkins 2017). In the scope of this thesis, Jenkins is used to schedule and run jobs that

execute automated tests. This is done via Jenkins "master/slave" mode, where the testing workload is delegated to multiple "slave" nodes, allowing a single Jenkins installation to host a number of test execution machines. By using Cron expressions to schedule jobs, a timeframe can be defined for the test execution and if should there happen to be multiple jobs scheduled within the same timeframe, Jenkins organizes the jobs so that the tests can run without interruptions. By using Jenkins, testing can be conducted in non-working hours and the results will be ready for examination in the morning.

By default, Jenkins reports its job execution using three markers: successful, unstable and failed. In test automation, this information is insufficient as testers would like to know what exactly went wrong during testing. By installing the Robot Framework plugin, the more detailed Robot Framework report straight to Jenkins "master" machine can be obtained.

3.6 TestRail

TestRail is a modern, web-based test case management software aimed for QA and development teams. It helps teams to verify their product's functionality and requirements by offering an easy way to manage all details in a structured way, including preconditions, steps and expected results (TestRail – Features, 2017). TestRail has a clean and modern interface that helps users to get a clear view of the statuses of their tests at a glimpse (Figure 5).

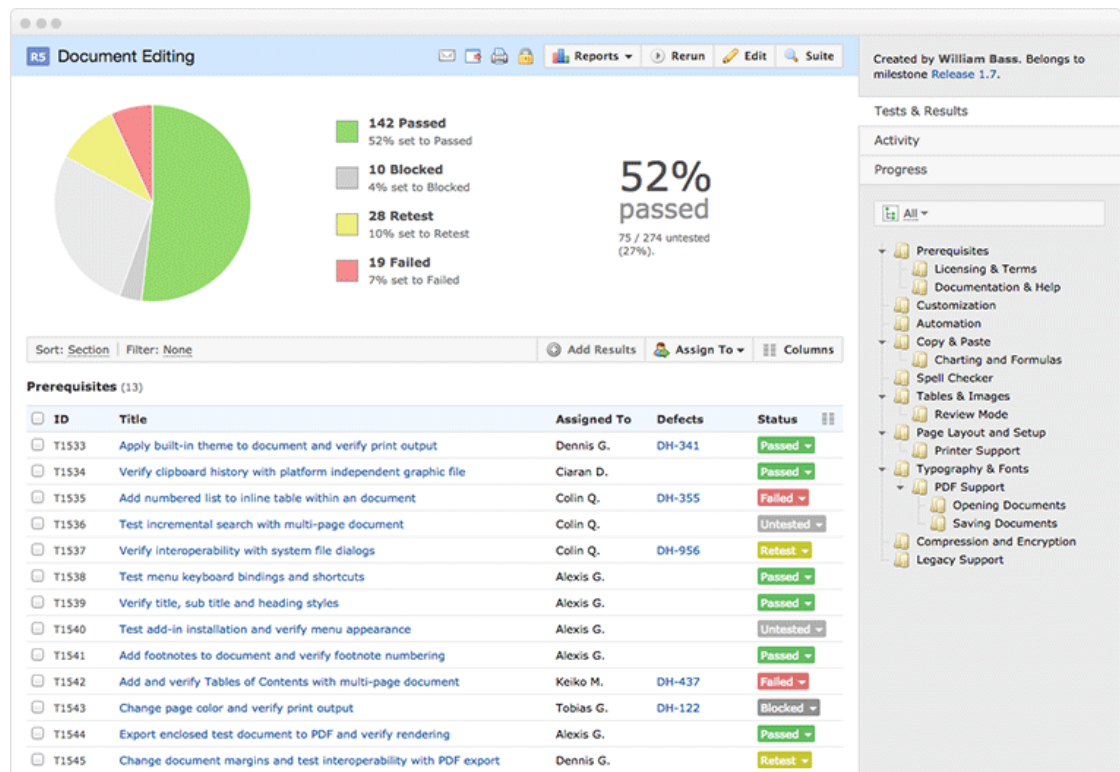


Figure 5. TestRail overview (TestRail, 2017)

With a complete JIRA integration, users can link JIRA stories and issues to specific test cases or test suites. The test execution statuses can then be viewed straight from JIRA, and possible defects and bugs can be pushed to JIRA without leaving TestRail. TestRail also offers an extensive API for integrating automated tests and submitting test results. With a custom built Python library for Robot Framework utilizing TestRail's API, it is extremely easy to automatically start a new test run, submit results and comments and to close the run after it is finished, so that the results cannot be altered afterwards. In the scope of this thesis, TestRail is a tool that offers a quick way to gain an understanding of the overall execution of the tests: how many tests have been run, how many have passed and how many have failed within a given timeframe.

3.7 MongoDB

MongoDB is an open-source cross-platform document-oriented NoSQL database program. It is aimed for storing large volumes of unstructured data in a JSON-like format (Figure 6) and is, at the time of writing, the leading NoSQL database.

```
{
  "_id": "4cc4a5c3f597e9db6e010109",
  "last_update": "2017-04-03 18:28",
  "users": [
    {
      "name": "Billy Bob",
      "age": 42
    },
    {
      "name": "Mary Lou",
      "age": 28
    }
  ],
  "tags": [ "people", "students" ]
}
```

Figure 6. MongoDB example document

Due to MongoDB's light weight, fast performance and flexible schemas, it was an ideal choice for storing performance data that can have substantial variations depending on the test case, unlike relational databases where a table's schema must be determined and declared before inserting the data. An application communicates with MongoDB by using a driver that handles all interaction with the database in a language appropriate to the application (MongoDB, 2017) or by REST API. In the scope of this thesis, both communication options were used: driver for inserting the data and REST API for retrieving it.

3.8 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment for developing scalable network tools and applications. It was designed to be an asynchronous environment that uses event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js also provides a variety of JavaScript modules that simplify the application development process to a great extent via its package ecosystem, npm, which is the largest ecosystem of open-source libraries in the world. (Node.js, 2017).

One of the reasons why Node.js is, at the time of writing, such a popular environment for web application development, in addition to the fantastic package ecosystem and efficiency, is its simplicity and flexibility. Setting up a simple web server takes mere minutes and requires only a few lines of code. (Figure 7.)

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Figure 7. A simple Node.js web server (Node.js – About, 2017)

However, the ever increasing amount of new modules makes the Node.js ecosystem move constantly. New and "better" tools are released almost on a daily basis and the Node.js community is quick to switch its opinion on whether or not to use a certain module anymore. When the community switches to a new module, the development of the old module slows down or stops completely, causing deprecation and dependency issues for those who still use it. In order to keep a Node.js application up-to-date, developers must keep up with the current trends, which can be a gruesome task if the application in question is large and complex.

In the scope of this thesis, however, Node.js based back-end solution works perfectly as its main job is extremely simple: communicate with MongoDB and serve the data. This kind of functionality has very few dependencies for modules and a small code base, so the shifting trends within the Node.js community should cause little to no problems in the future.

3.9 C3.js

C3.js is a D3.js based reusable JavaScript chart library. It makes data visualization extremely easy by wrapping the code that is required to construct D3.js charts. C3.js assigns CSS classes to each element when the chart is generated, allowing extensive custom styling for the entire graph (Why C3, 2017). C3.js provides tools for generating a variety of unique graphs for all kinds of data with only a few lines of code (Figure 8).

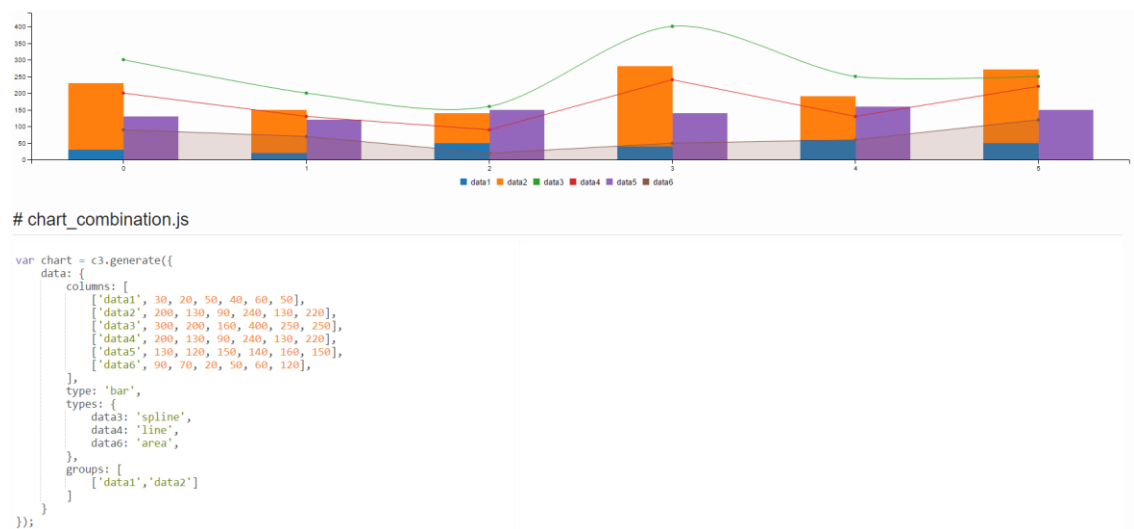


Figure 8. Combination chart sample (C3.js, 2017)

4 Workflow

4.1 Developing automated tests

4.1.1 Research

As with most cases when starting a new undertaking, the first step is to research and study the presented problem or task. Based on the author's own experiences, it can safely be said that understanding the problem before trying to solve it can drastically reduce the amount of unnecessary work. As high-level test automation is, practically speaking, just a computer mimicking the actions of the user, it is crucial for the developer to understand how the user performs that specific action.

In this case, the correct route to fully understanding the problem usually involves several manual executions of the given test case. However, before the test can be manually executed, each step required to perform it must be known. The best way to list these steps is to have someone familiar with the given test case to perform it manually, while the developer memorizes or writes down every single step. It is easy to see how this method automatically breaks a large and complicated test case into a smaller, more manageable pieces (Figure 9).

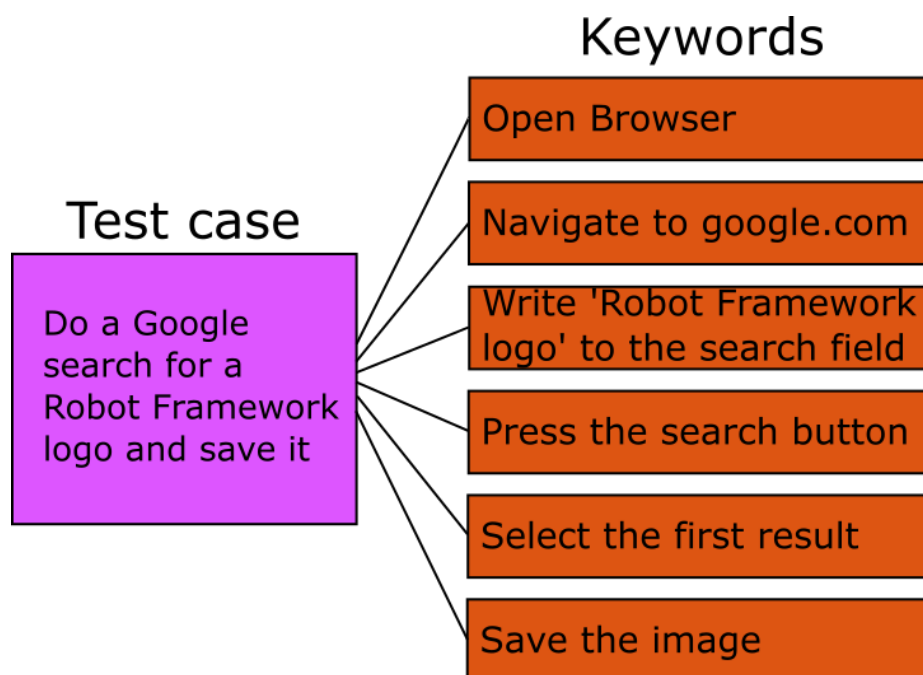


Figure 9. Test case breakdown example

The case now having been broken down to several small steps such as opening an application and inserting a text to a field or pressing a specific button, there is a clear view on how the actual execution takes place and automating it can be started.

4.1.2 Development

The following section demonstrates some good practices on starting a new Robot Framework project. These practices are derived from the author's own experiences and they are meant as helpful guidelines.

Folder structure

Like in traditional software development, logical and easy-to-understand folder structure makes a project much easier to maintain. Therefore, the first thing to do when starting a new project is to design and create an appropriate folder structure for the project. Each piece of the test, e.g. argument files, environment specific configuration files and keyword files, should have a dedicated folder (Figure 10).

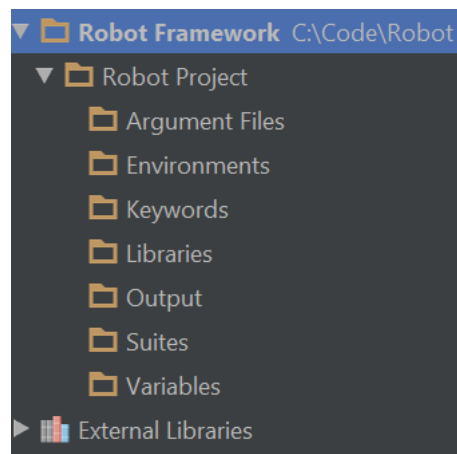


Figure 10. Folder structure

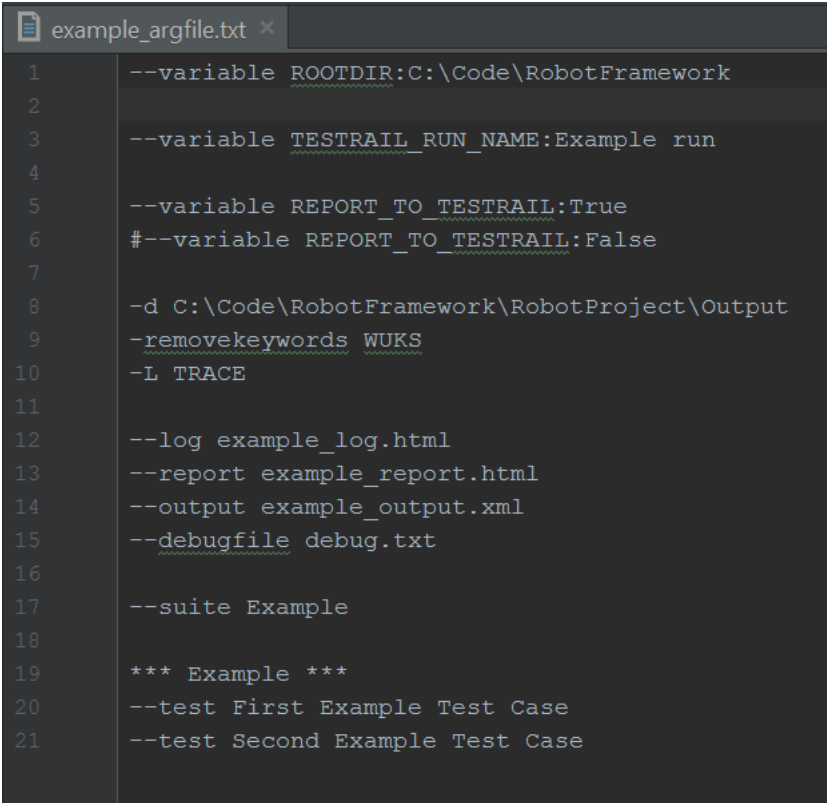
Imports

Once the folder structure is deemed appropriate, it is time to start thinking about what libraries and resources are needed for the given project. Is the system under test a Java-based system? Importing SwingLibrary might be a good idea. Is it a web application? Selenium2Library is a good option. Does the test include string

handling? Import String library. Larger projects may depend on multiple libraries and resources so importing them at a suite- or keyword level might cause the files to become cluttered. Therefore, a separate file for imports is a good solution for keeping the files nice and clean.

Argument file

Robot Framework tests can be executed straight from the IDE's terminal, however, large and complicated test suites may require several startup arguments and configurations which can make writing the startup command a fairly tedious task. Therefore, writing a batch file that uses a separate argument file to define the execution parameters is a good solution. Argument files can be used to define where Robot Framework should create the test logs, how to name them and what tests to execute, among other things (Figure 11).

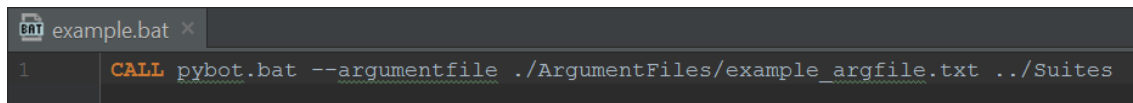


```
1  --variable ROOTDIR:C:\Code\RobotFramework
2
3  --variable TESTRAIL_RUN_NAME:Example run
4
5  --variable REPORT_TO_TESTRAIL:True
6  #--variable REPORT_TO_TESTRAIL:False
7
8  -d C:\Code\RobotFramework\RobotProject\Output
9  -removekeywords WUKS
10 -L TRACE
11
12 --log example_log.html
13 --report example_report.html
14 --output example_output.xml
15 --debugfile debug.txt
16
17 --suite Example
18
19 *** Example ***
20 --test First Example Test Case
21 --test Second Example Test Case
```

Figure 11. Argument file example

A few things to note about the argument file example are the flags -d and -L which define the desired output folder and logging level, respectively. In addition, the -

removekeywords flag can help to reduce the file size of the reports, especially in this example case where the extra Wait Until Keyword Succeeds -keywords are removed from the report and only the last one, whether it succeeded or not, is kept. Keeping the arguments in a separate file greatly simplifies the test configuration process and execution, as variables and tests can be commented out based on the current need and the test can be executed from the command line by running a simple batch script (Figure 12).



```
BAT example.bat x
1 CALL pybot.bat --argumentfile ./ArgumentFiles/example_argfile.txt ../Suites
```

Figure 12. Batch script example

Writing the test cases

After the project is correctly structured and all the relevant files have been created, writing the tests can begin. One of the ways to start developing test cases is to write the steps down chronologically at a suite level in order to get a view of logical keyword groups that can be wrapped under a new keyword. Once these groups are found, they are easy to wrap and move to a separate keyword file while maintaining an appropriate abstraction level.

The test under development should be executed constantly in order to spot and handle the problematic parts of the execution in time. In addition, checks to make sure that the test execution as well as the system under test are in correct and expected state, should be made regularly.

Once the test case is deemed as working, the last steps are sending the measured data to the database and writing the TestRail report. The TestRail report will always be written regardless of the test execution status, so a natural place for it is the teardown section of the case; unlike the database update part which might not be a good idea if the test execution fails for one reason or another. However, this functionality depends on the given test case.

4.1.3 Review

After a test case is deemed finished and working by the developer, it is time for a review. First is the code review which takes place via Gerrit. When the code is pushed to Gerrit it gets reviewed for logical errors and bad practices by other test automation developers. However, Gerrit is not used just for review purposes, as it also helps developers to stay on track what other developers are working on and possibly learn something when reading each other's code. Once the code is deemed appropriate and error free, it is accepted and pushed to the master branch.

There should be another review session in appropriate intervals with the party that had given the assignment in the first place. However, in this review session the main attraction is the logic of the test case, not the code itself. It should be confirmed that the test does what it should, collects the desired data and reports the results correctly. Once all the relevant parties are happy with the results, it is safe to consider the test development as finished and move on with the next case.

4.2 Data gathering and storing

The gathered data mostly consists of the execution time of a specific operation on the user interface and the amount of processed database rows. The execution time is measured as follows: once the test has reached the point of interest, the current time is stored in a variable with an accuracy of one millisecond. Using the Wait Until Keyword Succeeds -keyword, the automation observes the user interface in one millisecond intervals for a sign that the operation has finished. Once the operation is deemed as finished, the current time is once again stored in a variable and the total time for the given operation is calculated from the difference. If the operation failed for some reason, the test execution is aborted and marked as failed. If the operation finished successfully, the amount of processed database rows is read from the user interface, when available, and stored in a variable. Once all the data is gathered, a predefined database query string is updated using said data and a new database record is sent to the database as a last step before the test teardown procedures.

4.3 Displaying the results

Fetching the measured performance data from the database happens via REST API created with the Express framework for Node.js. Only two routes are defined in Node.js, both of which are GET method routes. There was no need for other HTTP operations, such as POST, PUT and DELETE, as the data is stored to the database directly from Robot Framework and subsequent data manipulation is not allowed. Therefore, the only required functionalities for the web server are to serve static web pages and retrieve the desired data that is displayed on those pages.

Each test case has a designated web page for data visualization which contains a preconfigured graph template for the data of that particular test case. When a page is loaded, a request containing the name of that particular test case is sent to the web server and the identifier of every test environment that that test case has been executed against is returned. A dropdown menu is then populated with those environments in order to easily switch between the results of different environments. Once the environments are fetched, another request containing the test's name and the identifier of the first environment that was just returned, is sent to the server. This request returns all the gathered data for that particular test case against that particular environment in a JSON format. The JSON containing the data is then passed on to the graph generation function which generates a graph based on the values of the JSON. For readability, only the 10 most recent measurements are displayed on the graph by default, however, a zoomable sub-chart that contains all the gathered data is positioned under the main graph.

Each data visualization page also has a TestRail link to the most recent test runs, as well as a badge that shows the latest Jenkins build status for that particular test case. The badge also links to the Jenkins job for that particular test case so it is easy to go analyze the Robot Framework logs straight from the data visualization page.

4.4 Automated test execution

4.4.1 Jenkins plugins

Jenkins installation is a fairly straightforward process of downloading and running the installer and starting the server. However, there are a few ease-of-life plugins available for cases where Jenkins is used to run Robot Framework test automation. First, the Robot Framework Plugin, could be considered mandatory as it allows uploading the test reports directly to Jenkins, defining the percentage of test cases required to pass in order to mark the suite execution successful, as well as displaying test execution trend graphs (Figure 13).

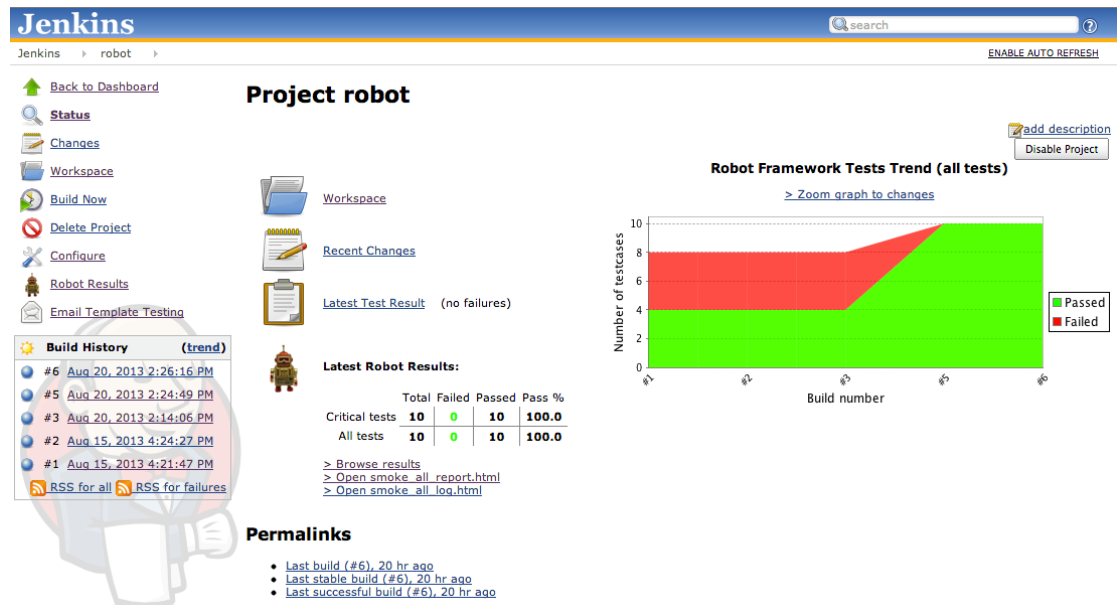


Figure 13. Jenkins Robot Framework Plugin (Robot Framework Plugin, 2016)

The second plugin, the NodeLabel Parameter Plugin, is an extremely useful plugin when dealing with multiple slave machines. It adds two new parameter types to job configuration – node and label, which allows dynamic selection of the node where a job should be executed. This functionality can be used to verify that a particular slave machine is correctly set up and ready for work.

The third plugin, the Build Monitor Plugin, is not mandatory by any means. However, it is a helpful tool for getting a clear overall view of all the Jenkins jobs that are being executed (Figure 14). It shows, for example, job execution statuses in real-time and

how long each job execution has lasted. The Build Monitor Plugin can be very informative when used in an open space where all relevant parties can get a quick glimpse about a project's overall status.

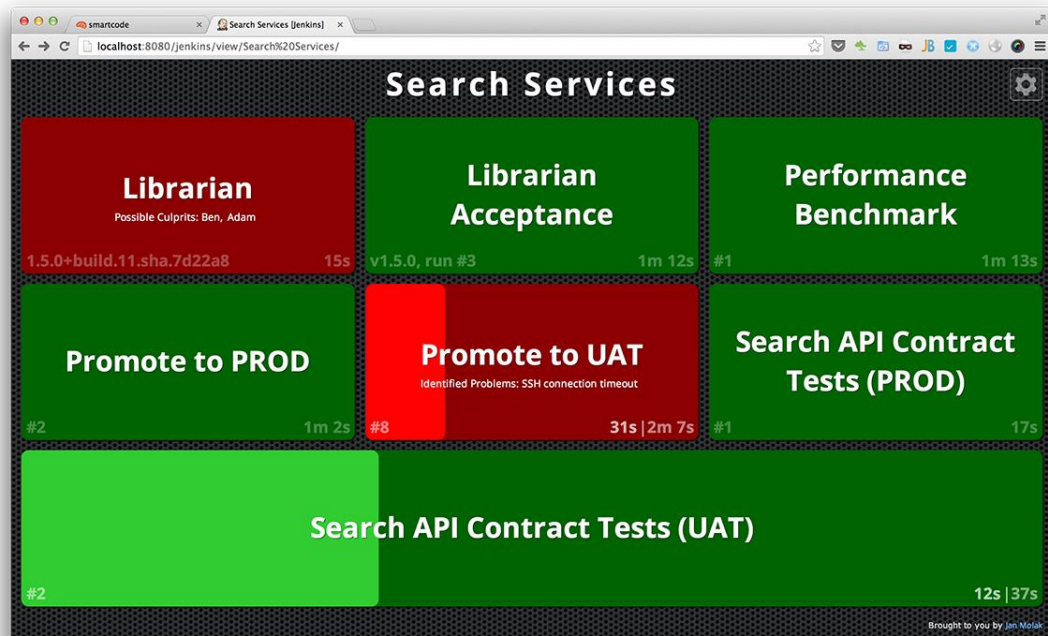


Figure 14. Jenkins Build Monitor Plugin (Build Monitor Plugin, 2017)

4.4.2 Slave machine setup

Creating a new Jenkins slave machine, or Node, happens through Manage Jenkins -> Manage Nodes menu. When creating a new slave machine there are few things that need setting up before the slave machine can be issued with jobs:

- Naming the slave machine
- Defining number of executors (how many concurrent jobs can be executed on a slave machine at a given time). A safe choice is one executor per slave machine as it prevents conflicts between processes run at the same time.
- Setting a Remote FS Root which is the Jenkins home directory on the slave machine (for Windows slaves, usually "C:\Jenkins\").
- Defining an appropriate Usage; *Utilize this slave as much as possible* and *Leave this machine for tied jobs only*, meaning that the slave machine can be

used to execute any jobs whenever the machine is in idle state or that the slave machine can only be used to execute specified jobs, respectively.


- Selecting a Launch Method for the slave machine. For Windows slaves, the recommended method is *Launch slave agents via Java Web Start*.
- Selecting the Availability option. In most cases the *Keep this slave on-line as much as possible* -option is appropriate.

After setting up the slave machine, the only thing to do is to connect the slave machine to the Jenkins master. There are a few different launch options available (Figure 15).



Slave Jenkins Slave

Connect slave to Jenkins one of these ways:

-  **Launch** Launch agent from browser on slave
- Run from slave command line:

```
javaws http://hudson-server/computer/Jenkins Slave/slave-agent.jnlp
```
- Or if the slave is headless:

```
java -jar slave.jar -jnlpUrl http://hudson-server/computer/Jenkins Slave/slave-agent.jnlp
```

Figure 15. Jenkins Slave Launch Options (Jenkins 2016)

Once the slave machine has been connected to the Jenkins master, it shows up as an available build executor with an idle status and the slave is now ready to start executing jobs.

4.4.3 Creating jobs

Jenkins job creation process is very flexible and allows configuration of every aspect of the build process. It is, for example, possible to define a parameterized job which allows the definition of certain environmental variables e.g. the environment to execute the test cases against and what slave machine the job should be executed with. Other notable configurations are the timeout limit and actions to take when that limit is reached.

Job scheduling is also done in this part. By checking the Build periodically -checkbox the job is defined as scheduled and the scheduling itself is done using Cron expressions (Figure 16).

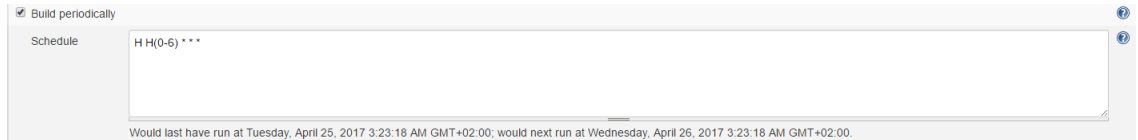


Figure 16. Job scheduling example

In the above example, the job is scheduled to be executed every night between midnight and 6 am. Jenkins keeps track on all the scheduled jobs and allocates a suitable timeframe for each job so that there are no conflicts.

In the build section of the job creation form, it is possible to add build steps to execute. In the case of executing Robot Framework tests in a Windows environment, a batch script that executes the desired test suite can be written. If the project is structured as mentioned in chapter 4.1.2, a batch script that executes a test suite is already written and the only required build step is to call that script.

The last option to define for a new job are the post-build actions. These actions are executed after the job has finished. The Robot Framework Plugin for Jenkins allows defining an action called Publish Robot Framework test results, which fetches the results files from a defined path and publishes them to Jenkins. The plugin expects a default naming for the results files (i.e. report.html and log.html), however, custom file names can be defined using the Advanced button and writing the desired names in relevant fields.

Now that the job is properly configured, the only thing left to do is save the job and it can then be executed from the Jenkins master.

5 Evaluation

At the time of writing, the developed test cases have been running nightly for a few weeks with a 97% pass rate. The data is properly collected and stored in the database, from which it then gets served to the data visualization pages. The whole project is built so that it is extremely simple to add new test cases. The only thing to do after a new test case is developed is to copy an existing visualization page, configure the graph as desired and define the name of the new test case. Using dynamic routing, the desired data is easily fetched using only an identifier for the test, which in this case is the test's name. The benefits of dynamic routing can also be seen when the tests are executed against a new environment. The new environment gets automatically added to the environment selection menu and the data is instantly available without any extra configurations.

Overall, the tests seem to be very stable and as the whole chain from periodic test execution to data storing, reporting and data visualization is fully automated, there is little to no need for maintenance. The full architecture for the project is available in the figure below (Figure 17).

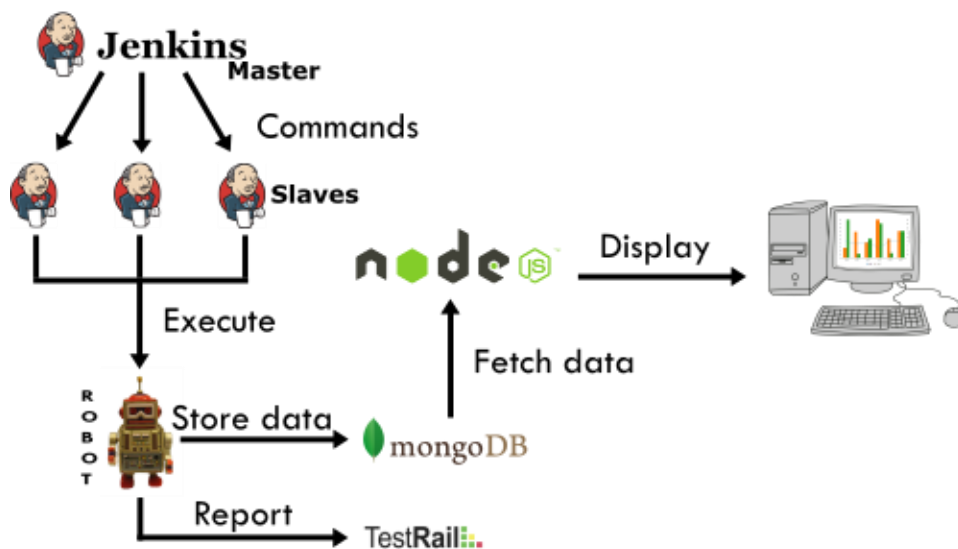


Figure 17. Project's overall architecture

6 Further development

At the time of writing, the required test cases are ready and the main focus of the project has been moved to setting up the final performance testing environment. Once the environment is configured properly, it can be defined as the testing target in Jenkins and the automation can start gathering performance data from the final testing environment. As adding new test cases for performance data gathering purposes is fairly simple, it is also possible that a need arises for a new set of test cases.

Improving filtering options for the fetched data is also on the list. As the amount of collected data grows bigger over time, the ability to filter the data by date or by the version of the system under test becomes almost mandatory. At the time of writing, the only data filtering option is the ability to define the environment the tests have been executed against. In order to keep the graphs readable, currently only 10 most recent test results are displayed by default and a sub-chart which includes all the gathered data is positioned under the main graph. Although it is possible to review all of the collected data by zooming around the sub-chart, a more flexible and user-friendly solution is definitely needed.

7 Conclusion

All the required performance test cases were designed and developed successfully and the solution for automatic test execution, reporting, data gathering and data visualization works well. The test development process greatly enhanced my skills as a test automation developer and due to the need for testing library enhancements some experience of writing custom library additions in both Python and Java was also gained. A well designed overall solution makes further development extremely easy. New test cases can be added to the solution with only a minimal amount of configuration.

Prior knowledge of open-source technologies like Node.js, Express.js and MongoDB made the back-end development process extremely fast. With only under hundred lines of code, a simple web-server that communicates with the database and serves static web pages was fully working and stable. By using these light-weight technologies the server experiences only minimal load and requires little to no maintenance.

When it comes to developing test automation, Robot Framework is an excellent tool. By following simple guidelines, it is possible to create test cases that excel in readability and maintainability. Large variety of different testing libraries enables test automation development regardless of the programming language the system under test is based on. By combining Robot Framework and Jenkins automation server with open-source back-end technologies, a testing solution that is both reliable and self-sufficient can be created easily.

The objectives of the thesis were met and the performance test suites are ready to be deployed to the actual testing environment. The created solution makes monitoring the performance of specific graphical user interface functions simple and user friendly. By constant performance monitoring, the chances of uncovering new software bugs and regressions are greatly increased, which reduces the need for repetitive manual labor.

References

Atlassian – Agile 2017. Accessed on 28.03.2017. Retrieved from

<https://www.atlassian.com/software/jira/agile/>.

Atlassian – Customers 2017. Accessed on 28.03.2017. Retrieved from

<https://www.atlassian.com/customers/>.

Atlassian – Delivery Vehicles 2017. Accessed on 28.03.2017. Retrieved from

<https://www.atlassian.com/agile/delivery-vehicles/>.

C3.js – Combination Chart 2017. Accessed on 26.04.2017. Retrieved from

http://c3js.org/samples/chart_combination.html/.

C3.js – Why C3? 2017. Accessed on 26.04.2017. Retrieved from <http://c3js.org/>.

Gerrit Code Review – A Quick Introduction 2017. Accessed on 21.03.2017. Retrieved

from <https://review.openstack.org/Documentation/intro-quick.html/>.

Gerrit Code Review 2017. Accessed on 21.03.2017. Retrieved from

<https://www.gerritcodereview.com/>.

Gerrit Workflow 2014. Accessed on 21.03.2017. Retrieved from

<http://blogs.wandisco.com/2014/09/22/gerrit-workflow/>.

Jenkins 2017. Accessed on 28.04.2017. Retrieved from <https://jenkins.io/>.

Jenkins - Build Monitor Plugin 2017. Accessed on 28.04.2017. Retrieved from

<https://wiki.jenkins-ci.org/display/JENKINS/Build+Monitor+Plugin/>.

Jenkins - Robot Framework Plugin 2016. Accessed on 28.04.2017. Retrieved from

<https://wiki.jenkins-ci.org/display/JENKINS/Robot+Framework+Plugin/>.

Jenkins – Step by step guide to set up master and slave machines 2016. Accessed on

01.05.2017. Retrieved from <https://wiki.jenkins-ci.org/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machine+on+Windows/>.

Kumar, A. 2015. Git Branching Strategies. Accessed on 20.03.2017. Retrieved from

<https://www.javacodegeeks.com/2015/11/git-branching-strategies.html/>.

Landis-Gyr - Gridstream AIM 2017. Accessed on 17.04.2017. Retrieved from

<http://www.landisgyr.fi/product/gridstream-aim/>.

Landis+Gyr: A Smart Meter Pioneer 2017. Accessed on 16.03.2017. Retrieved from <http://www.landisgyr.com/>.

MongoDB Drivers and Client Libraries 2017. Accessed on 03.04.2017. Retrieved from <https://docs.mongodb.com/manual/application/drivers/>.

Node.js 2017. Accessed on 06.04.2017. Retrieved from <https://nodejs.org/>.

Node.js – About 2017. Accessed on 06.04.2017. Retrieved from <https://nodejs.org/en/about/>.

Klärck, P. 2014. Robot Framework Dos and Don'ts. Accessed on 15.05.2017. Retrieved from <https://www.slideshare.net/pekkaklarck/robot-framework-dos-and-donts/>.

Robot Framework - Introduction 2017. Accessed on 11.04.2017. Retrieved from <http://robotframework.org/>.

Robot Framework User Guide 2017. Accessed on 11.04.2017. Retrieved from <http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html/>.

TestRail – Features 2017. Accessed on 29.03.2017. Retrieved from <http://www.gurock.com/testrail/tour/modern-test-management/>.

What is Performance testing in software? 2017. Accessed on 16.03.2017. Retrieved from <http://istqbexamcertification.com/what-is-performance-testing-in-software/>.