Haaga-Helia
ammattikorkeakoulu Oy

Improving data analysis in continuous integration environment

Joni Taajamo

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2020

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Author(s)
Joni Taajamo

Degree programme
Tietojenkäsittelyn koulutusohjelma / Business Information Technology

Report/thesis title
Improving data analysis in continuous integration environment

Number of pages and appendix pages
31 + 5

This thesis reports the process of developing a data pipeline system for Nokia Networks. The main goal was to plan, design and develop a data analytics system that analyzes the test results from the continuous integration (CI) system. Requirements for the system were gathered from stakeholders and it was developed using a Lean method, Scrum. The project was conducted during Q1 of 2020.

This report begins by demonstrating the research and benchmarking of relevant technologies and methods. The system's key technologies are Python, Microsoft Power BI, Docker, PostgreSQL and MongoDB. Some of the technologies were used at the request of Nokia Networks. This phase helped to design the final system and to confirm the selected technologies.

After the research phase, details of the implementation are discussed. Most parts of the system are confidential to Nokia Networks, but technical aspects are presented using examples. The system is broadly formed of two separate parts: a data pipeline which migrates the data from original source to analyzable form and the report which visualizes the results. Implementation was successfully developed and deployed into production.

Finally, the results of the project are evaluated with a brief technical review and user satisfaction inquiry. The results from the inquiry were positive and verified the quality of the system. Projects goals of reducing the time of test analysis and inspiring other projects were reached. The report is concluded by discussing the key learnings from the project.

Keywords
analytics, automation, processing, testing

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Joni Taajamo

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Data-analytiikan parantaminen jatkuvan integraation ympäristössä

Sivu- ja liitesivumäärä
31 + 5

Tässä opinnäytetyöraportissa käydään läpi dataa automaattisesti analysoivan järjestelmän kehitysvaiheet. Opinnäytetyön tavoite oli suunnitella ja toteuttaa dataa analysoiva järjestelmä, joka analysoi jatkuvassa integraatiossa toteutettujen testien tuloksia toimeksiantajana toimivan Nokia Networks-yrityksen yksikössä. Projekti toteutettiin käyttäen ketterää menetelmää, Scrumia. Projekti toteutettiin vuoden 2020 ensimmäisellä neljänneksellä.

Raportin alkuvaiheessa esitellään ja vertaillaan projektin toteutukseen liittyviä teknologioita. Järjestelmän tärkeimmät teknologiat olivat Python, Microsoft Power BI, Docker, PostgreSQL ja MongoDB. Osaa teknologioista, kuten Power BI:tä käytettiin toimeksiantajan toiveesta. Tämän vaiheen tarkoituksena on auttaa varsinaisessa kehityksessä ja vahvistamaan käytettyjen teknologioiden ymmärrystä.

Tietoperustan jälkeen käsitellään varsinaista järjestelmän toteutusta. Suurin osa toteutuksesta on salassa pidettävää, mutta yksityiskohtia esitellään esimerkkien avulla. Järjestelmän toteutus voidaan jakaa karkeasti kahteen osaan: datan käsiteltävään muotoon muuttava osuus ja datan visualisoiva raportti. Toteutus oli onnistunut ja se on viety tuotantoon.

Raportin loppupuolella käsitellään projektin tuloksia lyhyen teknisen arvioinnin ja käyttäjätyytyväisyyskyselyn avulla. Käyttäjäkyselyn tulokset olivat positiivisia. Käyttäjätulosten perusteella projekti saavutti tavoitteensa, jotka olivat manuaalisen työn vähentäminen ja muiden vastaavien projektien innoittaminen. Raportin lopuksi käsitellään siitä opittuja asioita, joita oli runsaasti.

Asiasanat
analytiikka, automaatio, prosessointi, testaus

Table of contents

1 Introduction

This thesis is commissioned by Nokia Networks to implement a data analysis system for one of their product's Continuous Integration system. First, related technologies are researched in this report, then implementation is discussed, finally results of the thesis are reported. This thesis report is generalized in some parts due to confidential information; no real data or information from any internal systems is represented in the thesis or its attachments.

1.1 Background

Today, organizations require fast response times to stay relevant when developing software, testing and deploying it even every 12 seconds (Aijaz 2019). This means that all components of software development teams must work in cohesion to reduce time to deploy and every step after programming must be automated. Multiple principles and practices have been introduced to meet the requirements of modern teams, such as DevOps. (Kim, Humble, Debois & Willis 2016, 3.) Faster deployments come with an increased need for data analysis to detect difficulties with deployment pipelines and test environments. This thesis focuses on improving data analysis in a Product Continuous Integration (PCI) team inside Nokia Networks. The responsibilities of the PCI team include carrying out regression testing and integrating networking software before it's deployed to production.

1.2 Objectives

Massive amounts of data is generated from continuously testing and integrating software. In the customer PCI team, the data is saved to a database after each build where it can be accessed and analyzed. Currently, the only form of access to all data is through raw JSON-documents (JavaScript Object Notation). It is not an efficient way to quickly analyze the results. The PCI team had an old desktop system for analysis, but it needs to be updated and redevelopment is sensible with more established technologies

The objective of this thesis is to plan and build a visualization tool and a data pipeline for the PCI team inside Nokia Networks. Data needs to be available in visualized format, in almost real-time. New charts from the data could be created easily, even without special knowledge. This will be achieved by formatting the data into a relational model and feeding it into a visualization system, in this case, Microsoft Power BI. Required software components for data formatting need to follow best practices and be well documented for further development and maintainability.

By providing an analysis tool, the work of the PCI team will be streamlined, and many workhours could be saved. Also, this work will be a catalyst for building more automation and analysis tools for Continuous Integration. During the development of the data pipeline and analysis tool, problems with the current solutions may be found. These problems should be considered with the team and in this thesis.

This project should also act as a learning process for me, the thesis author. Everything learnt when doing the thesis should be applicable to later projects.

1.3 Nokia Networks

Nokia Networks is Nokia Corporation's business unit, focused mainly on developing mobile networking technologies. The company's portfolio includes providing services to mobile access service providers around the world and its current focal point is in developing, researching and deploying 5G networks. (Nokia Oyj 2020) This thesis is conducted for Nokia Networks and the writer is a salaried employee there.

2 Continuous Integration

Continuous Integration or CI is today's standard what it comes to testing and deploying software into the production environment (Fowler 2006). Programming is just the first part of process of software development. After that, it needs to be tested on many levels, deployed to a suitable environment and then monitored. CI tries to streamline and formalize this process.

2.1 Method and history

The term Continuous Integration was first proposed by Kent Beck in his book 'Extreme Programming' (Fowler 2006). It refers to the process of testing and deploying software to a state, where it can be deployed instantly. It is close to the term Continuous Deployment (CD), but they should not be confused. Both are referred as CI/CD together when talking about modern testing and deployment automation, but they do not mean the same thing. CD happens after CI, if an organization chooses to do so: in CD software is continuously deployed to the production environment after previous phases have succeeded.

Historically, integrating and deploying the software has been the responsibility of an operations team without much automation (Kim et al. 2016. 0.). The development team just handed the code over to the operations team, who took care of installing all the required environments and hardware. The DevOps movement originates from the Lean principles which aim to remove inefficiency from manufacturing (Kim et al. 2016. 7.). CI is achieved by automating each step in the process and not doing them by hand (figure 1).

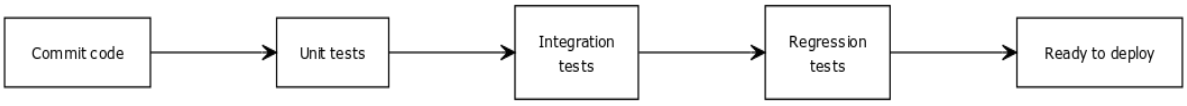


Figure 1 Continuous Integration steps

2.2 Tools

To automate each step of CI efficiently, a set of modern tools is required. These tools include scripting languages, test automation tools, networking managers and virtualization technologies. Tools vary greatly from organization to organization and tools are offered by many companies from open source products to closed-source, hosted services. Usually the implementation using these tools is referred as deployment or integration pipeline, an automated process to compile, test on different levels and prepare the software for deployment (Humble & Farley 2010).

In the following section, tools are introduced based on their relevancy to implementing the thesis. Nokia and many other companies use these technologies to implement CI/CD pipelines, but many alternatives exist.

2.2.1 Jenkins

Jenkins is an open source automation server that can be installed on-premises by any organization for free. It can automate many different tasks related to continuous integration including building, testing and deploying software. Jenkins is extremely flexible and customizable with plugins that can be developed by organizations for their special needs (Jenkins 2020). Some of the largest competitors for Jenkins are CircleCI and Travis CI who offer their services as paid plans (CircleCI 2020; Travis CI 2020). Jenkins is attractive to businesses because of its high customizability and open source code, which these competitors do not offer.

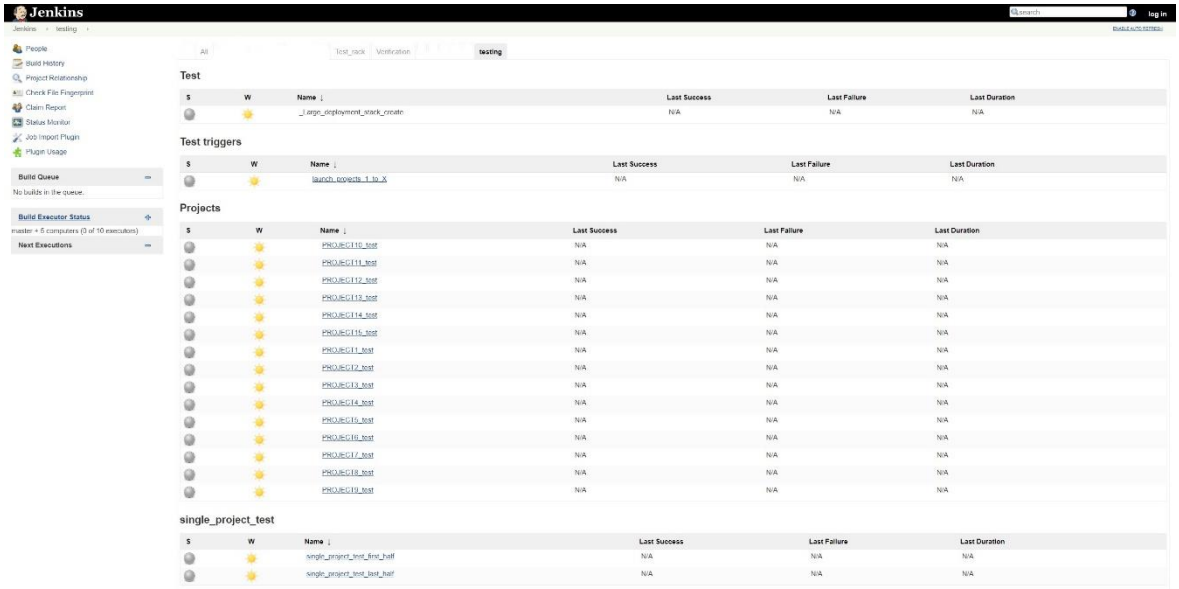


Figure 2 Example of Jenkins main view

Each automation project is referred as a Jenkins job, result of one run of a job is referred as a build. When describing builds, usually they can be successful or failed. The one who analyses these builds usually studies why the build has failed by inspecting some of the artifacts, or logs which the job has been configured to save after completed build. (Jenkins 2017) Jenkins is used at Nokia Networks for CI/CD-automation.

2.2.2 Robot Framework

Robot Framework is a keyword-driven open source automation framework used of acceptance testing among many other things, such as Robotic Process Automation. (Robot

Framework 2020) Robot Framework was initially developed at Nokia and was later used and developed by many other multi-national corporations for automated testing purposes. The language offers high customizability for each purpose with keywords. Any errors occurring during testing can be analyzed with its well-integrated reporting features.

2.2.3 Containerization

Containerization is a common trend in today's software development (IBM 2019). Containers are so called lightweight virtual machines, that mimic a "real" operating system for the containerized software and no full operating system is required. It allows software to be portable and be to run anywhere where the underlying container engine can be run, such as Docker Engine.

Containers allow the required environment to be described as parameters which are then automatically built by the container engine. Thus, containerization can be considered an enabler for Continuous Integration, as environment problems are less likely to happen during automation and resources can be reserved as needed.

2.2.4 Python 3

Python is an interpreted and dynamically typed programming language (Google for Education). It allows programmers to do a lot using only few lines of code. Obviously, this does not come without a downside: as python is an interpreted language it is slower to execute than languages like C or C++ (Lambert 2017). While it may be slow, it is not drastically slower in applications that do not require complex calculations. Python's default regular expression library even beats Java's or Golang's counterpart (The Computer Language Benchmarks Game). Regular expressions are often required in applications such as automation, where data needs to be parsed to process it. Python is often used for build control and management, so it is a good option for CI purposes (Python). Python is extensible with libraries, which are managed using pip, Python's package manager (Google for Education).

3 Databases and data migration

Databases come in many different shapes and forms for different purposes. Most databases still follow the same basic principles: data is gathered from somewhere, inserted into the database where it can be accessed and modified from many different programming languages and clients.

3.1 Relational databases

A relational model is a means of describing data within its natural structure, without superimposing any additional structure for programming purposes (Codd 1970). Structured Query Language (SQL) is one of the most used technology for relational modeling nowadays. SQL was developed by IBM in 1970s in San José and it was one of the first commercially available query languages (Laine).

One of the main goals of relational databases is to keep the data in consistent form, usually by only performing ACID transactions (Haerder & Reuter 1983). ACID stands for the four properties that any relational database should follow. It is short for following:

- Atomicity: queries are “all-or-nothing”, and user always knows which state they are in.
- Consistency: each commit contains only predefined, legal transactions.
- Isolation: transactions are not aware of each other.
- Durability: all transactions are held after commit even if the system suffers an error.

Name	**Address**	**City**	**…**
Matti	Autokuja 1	Vantaa	…
Jaakko	Syystie 44	Helsinki	…
John	Doe rd. 13	San Fransisco	…
…	…	…	…

Table 1: Example of relational data table

For example, some modern relational database systems are PostgreSQL, SQLite and MySQL. All these systems support SQL as their query language. Most of these database systems need to follow their own rules in structuring and querying the database.

While needing to follow rules set by the SQL standard, databases should also follow rules of normalization to present the data in consistent form (Kent 1982). Normalization is separated to five levels in the model originally represented by William Kent. Simplified, each of these levels bring a new set of rules for the database: under the first normal form all occurrences of a record type must contain same number of fields; under the second and third normal forms each record needs separate non-related values to their own records;

under the fourth and fifth normal forms record should not include independent two or more fact about an entity and all the facts should be as small as possible. The goal of normalization is thus to represent as small pieces of independent information as possible inside a database, that can be related to each other.

In programming, several techniques can be used to handle the data queried from relational database. One of them is ORM (Object-Relational Mapping), that maps the fetched data to an easily accessible form, for example to a Java Object (Hibernate 2020). It helps with achieving persistent form for the data inside a program.

3.2 Non-relational databases

Modern technology has brought many alternatives for relational databases, such as NoSQL. NoSQL usually refers to a document database, such as MongoDB, that stores the data as JSON-like (JavaScript Object Notation) documents. In reality, MongoDB uses BSON-format or Binary JSON, that may look like JSON, but is stored differently in computer memory (MongoDB 2020d). JSON allows programmers to store data as it is handled by the program: as key-value pairs, and no strict schema is required (MongoDB 2020). While relational databases store the values in tables (e.g. table 1), in NoSQL database MongoDB the equivalent is referred as a collection (MongoDB 2020d). Naming may vary greatly between NoSQL database systems, because it is not standardized like SQL.

If the data's structure changes at any part of the project, the change can be easily implemented and no changes to database itself is necessarily needed. This freedom does not come without problems, as it allows drastic changes to the data structures by the programmers. If no strict rules are enforced when programming, the data may be in non-distinct form and cause unexpected behavior or performance problems. Some NoSQL systems try to counter this with retaining structure by enforcing rules while holding the fluidity of NoSQL systems (Tozzi 2016). Many business intelligence tools usually require the data in relational form, so the document-based data needs to be translated before use.

3.3 Data migration

Data migration refers to movement of data from system to another as one-off movement. It is required if data structure changes or system is changed to another. It may be resource heavy operation for businesses but required in some cases. Continuously transporting from database to another, for example, long-term storage, is not called migration. Migration is usually needed if new long-term storage is established and old data needs to

be moved there (Morris 2006). In this project, Python will be used to migrate and synchronize data from MongoDB to PostgreSQL to translate the data from JSON model to relational model.

4 Data analysis, data science and data visualization

Data science is modern statistics that has combined modern sciences, for example, computer science and information science (Wu 1997, 12). Data science is applied into a wide range of different technologies like machine learning, neural networks and computational statistics. In this thesis, the focus is on automatic visualization and reporting, which could also be part of the data science field.

4.1 Business Intelligence

In today's businesses, data analysis plays an important role in decision making. Many tools and models are used to try to predict the future with data or to compare company performance to previous time-periods or competitors in order to gain advantage over them. In the business context, data analysis is usually referred as "business intelligence" or BI (Dedić & Stanier 2017). BI is used to represent relevant data about many layers of the business such as sales, marketing, end-user satisfaction, market, share and technical aspects such as my use case: performance of continuous integration and test results. Here I try to gain knowledge that can be used to improve the quality of the software and even show problematic points in the deployment pipeline for the customer.

4.2 Data visualization models for analyzing continuous integration

Provided the right tools and correct graphics, an overview of continuous integration system overall performance should be achieved as easy as possible via visualized graphics. These graphics can be further analyzed by a tester to form an understanding of which testcases are most prone to errors and what caused the error. Later, any results from automatic analytics, such as machine learning, can be added to the visualizations.

To select correct visualization models, understanding what are the relevant datapoints is important. Attachment 2 can see an example of one build's JSON-formatted file. Each Jenkins build and its related testcases are stored into the Mongo DB database in this format. It is translated into relational format by migrating it to PostgreSQL database and further synchronized continuously with the MongoDB database.

One goal of this thesis is to compare testcase runs of a single testcase to its previous runs. Values that can be used to compare runs to each other are duration and status of the testcase: did it pass or fail. These can be easily represented as a list, but anomality detection would be difficult. To gain a better understanding of the data, it should be represented in graphical format (Spear 1952, 3). Pass and fail ratio can be represented as a pie

chart, duration as a line chart to display test's start time on x-axis and duration on y-axis. Line chart is usually used to represent related data to each other, here a trend line can be visualized and see if any anomalies can be detected in the runs. These values can then be filtered further by testcase run, Jenkins build, testcase owner, start time or status.

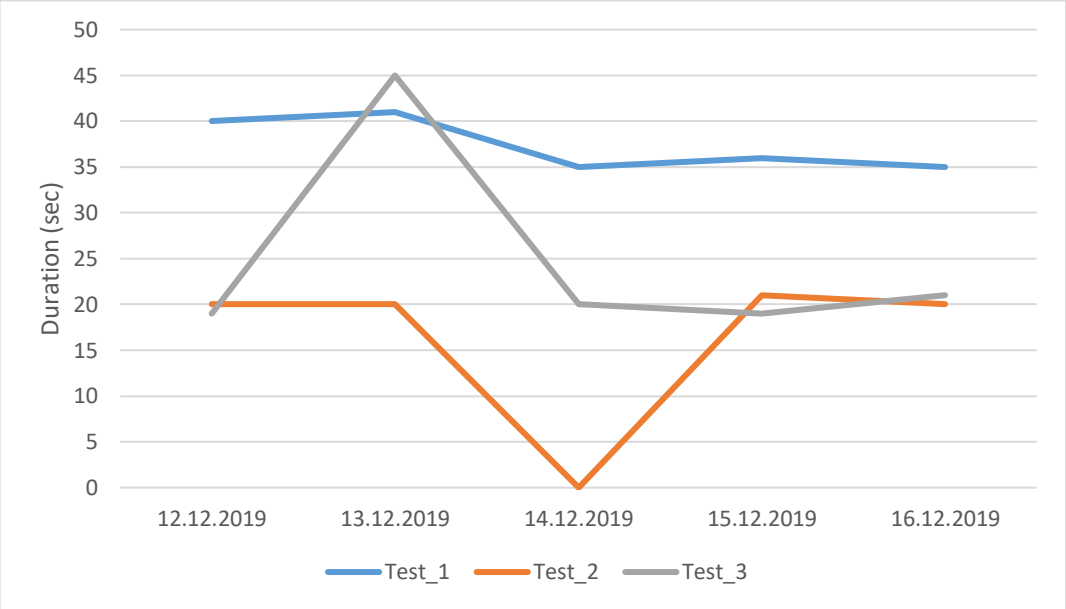


Figure 3 Example of a line chart.

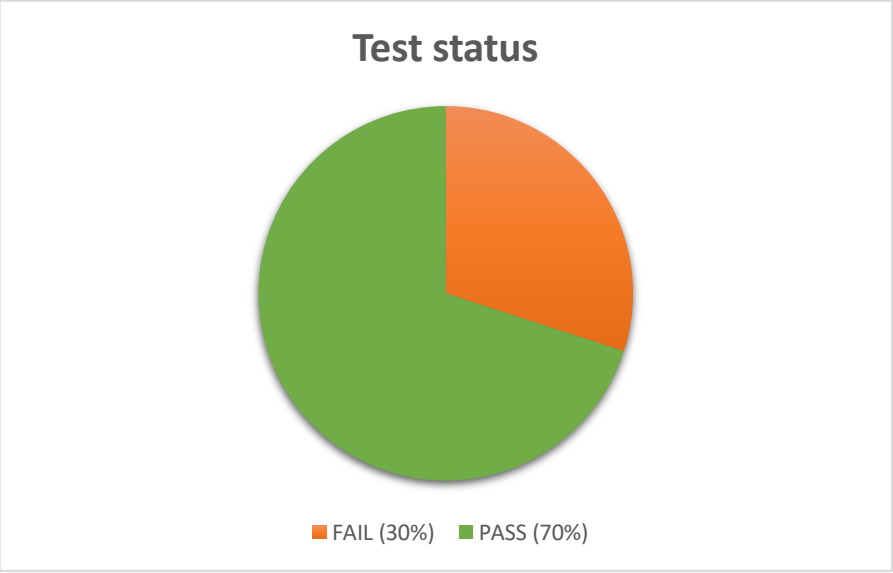


Figure 4 Example of a pie chart

In figure 3, values of test_2 and test_3 have detectable anomalies: Test_3 have taken longer than usual on 13.12 and test_2 have taken suspiciously short time on 14.12. By finding these occurrences, person who is analyzing the test results could easily look into them.

4.3 Microsoft Power BI

A data pipeline is the set of actions required to gather data from all the required sources for analysis and further processing (Marazzi 2016). Microsoft Power BI is one option for the last step of this pipeline.

Power BI is a business intelligence tool developed by Microsoft, released in 2016. It is a flexible platform that can be used to visualize and transform data from many different sources (Microsoft). Reports can be formed easily via premade modules and data points can be automatically detected. Adapted charts and modules can also be defined via tools provided by Microsoft or with Python or R. The software is still in active development and new features are added in every release.

This thesis project's user-facing end is developed using Power BI. Data is served to Power BI from PostgreSQL data source and Power BI's automatic relation finding features are used to map the data. One of the reasons this tool was selected is its' ease of use and easy development, if any changes are needed. Most of the selected (see 4.2) visualizations are readily available in Power BI.

Microsoft offers a Power BI desktop app, which can be used to develop the reports efficiently and do testing locally. They also provide a service, which allows to edit the reports online, but also share them with stakeholders and update the data sources on schedule. There are couple of different subscriptions offered to this service pro and premium, which unlocks more features, such as scheduled data source updates and more powerful computing services.

5 Thesis working method: Scrum

The thesis product is developed using scrum, an agile framework used by many software development teams today. It is derived from Lean principles. Scrum enables fast response times and agile changes to the development process and to the product's requirements. (Scrum 2020) It has been developed to address weaknesses with old waterfall model, where requirement specification was done once at the outset. After specification, the waterfall process went forward step by step and any changes to the product was hard to implement if problems with the requirements were found at a later step. Winston Royce (1970) described the waterfall model in an article where he described it as a flawed and risky model that should not be used, although he did not describe the model as a "waterfall" model.

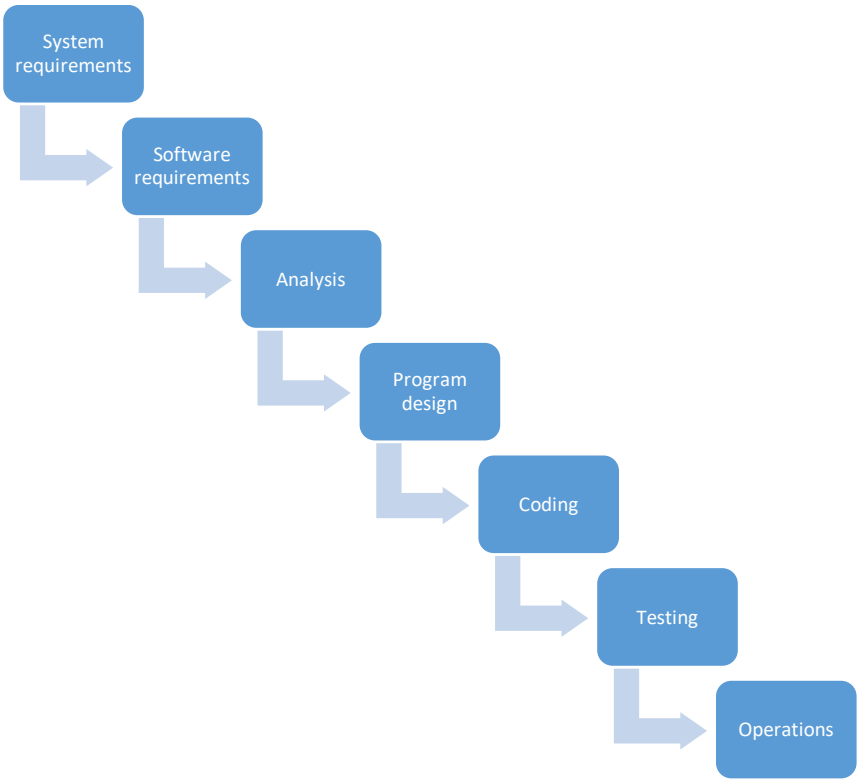


Figure 5 "Waterfall" model, development process (Royce 1970)

As mentioned in chapter 2.1, agile methodologies and CI/CD highly support each other.

5.1 Scrum roles

To apply scrum to a project, each stakeholder must have a defined role in the scrum project. The available roles are scrum master, product owner and development team. The scrum master is the one who manages the sprints and meetings (Scrum 2020). The scrum master is not a leader in traditional sense, they do not command the development

team or micromanage it. Usually, the scrum master is referred as a “servant leader”, leader who tries to ease work of the development team by removing blockers from development team’s daily work.

The product owner is responsible for setting the goals for the development team, but they do not part-take in deciding how the work should be done, only what stories should be done (Scrum 2020). They also ensure that everyone understands the backlog items completely and is the one who is most responsible for communicating with the customer and can even be employee of the customer.

The development team is responsible for turning the product backlog into a functioning product. It is usually formed of 3-10 members from different backgrounds. No titles for development team members are recognized (Scrum 2020). The team is cross-functional, meaning that skills may be split between the members, but the team as a whole is expert on the task at hand.

5.2 Scrum events

When developing products using Scrum, multiple events are iterated during sprints, starting from sprint planning. Each of these events have their own purpose to advance the project forward. Events are sprint, sprint planning, daily scrum, print review and sprint retrospective

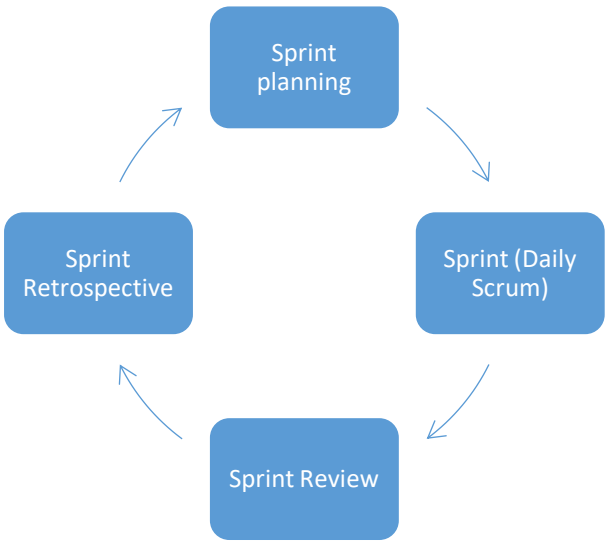


Figure 6 Scrum events (Scrum 2020)

Arguably the most important event for scrum is the sprint. It is a set length period, which includes all the waterfall model's steps. The sprint is then repeated, until the product is accepted by the customer (Scrum 2020). The usual length for sprints is 2-4 weeks.

Sprint planning is a meeting where all the work for sprint is planned. It has a set length of eight hours for one-month sprints. During the session, work is selected from the product backlog and listed into the sprint backlog (see 5.3) (Scrum 2020). Each listed job is given a value of its importance, usually a "story point".

Daily scrum is a 15-minute meeting where team synchronizes activities and creates a plan for next 24-hours (Scrum 2020). The scrum master ensures every scrum team member gets a turn to speak and no-one interrupts the speech. The daily scrum is repeated every day during a sprint.

The sprint review is held at the end of each sprint to inspect what work has been done and to adapt product backlog (Scrum 2020). It is usually a four-hour meeting when using one-month sprint length. During the sprint review the product itself can be demoed even to a low level.

The sprint retrospective occurs after scrum review and before sprint planning (Scrum 2020). In the retrospective, the scrum team inspects itself and creates a plan to improve their work to the next sprint.

Each of these events can vary slightly by scrum team, but they still need to follow the rule's set by the framework. For example, sprint planning and review can be arranged using many different methods, such as planning poker (Grenning 2020).

5.3 Scrum artifacts

Multiple artifacts are required to manage the scrum team and to arrange the work into sprints. Required artifacts are product and sprint backlogs. The poduct backlog is an ordered list about all the requirements that are needed for the product (Scrum 2020). It can be refined over time between sprints to match changed requirements (see 4.1.2 Sprint planning). The sprint backlog is a set of items selected from the product backlog to be completed during the sprint. It needs to be visible to all development team members.

5.4 Implementing Scrum in this project

All the functional parts of this thesis are developed following Scrum's principles. The customer for this project is the PCI team. I work in the development team alone, but I will be supported on technical and management matters by the customer scrum team.

Initial requirements for the software are collected from the older analysis tool and updated for current needs in a sprint planning session. Backlog items for the first sprint are listed on the product backlog (attachment 1) in user story form: "As a <role> I can <capability>, so that <benefit>". The backlog is refined between sprints based on findings during sprints and demo meetings with the customer. Product backlog items will be extracted into internal software called Atlassian Jira, which works as the sprint backlog. Jira allows the team to see how the development is progressing in a transparent way. Stories' and tasks' priority are defined by the customer and the work is planned with the customer.

Thesis product will be developed using two-week sprints and using guidance from scrum master. The product will be demoed to the customer team between each sprint iteration and the customer is free to adjust the requirements during the project.

Potential risks for the project are delays in getting required licenses for Microsoft products and getting access to required environments. To counter these risks, licenses and environments needs to be requested at the earliest stage possible. If there are delays in the requests, the project needs to be divided into parts that can be advanced separately.

6 Implementation

Implementation of the project started with gathering the requirements and form user stories from them. When starting the development sprints, a general design for the data pipeline is required. After the initial design, development of the migration tool with python can begin. Then developing reports from the migrated data can be started. These tasks can be worked on side by side and incrementally, but some steps require the previous step to function. After each sprint, something of value needs to be represented to the customer, so selection of tasks for each sprint is critical.

6.1 General data pipeline architecture and technologies

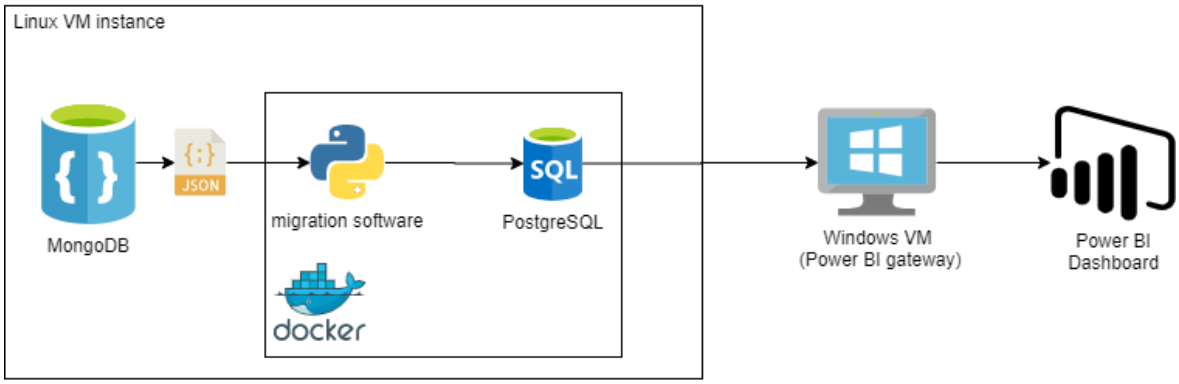


Figure 7 Architecture of the data pipeline

To bring the data all the way from MongoDB to the Power BI service, it needs to be first migrated and served through Microsoft Power BI gateway. The gateway allows to connect on-premises data sources securely to Power BI service, which is hosted in a public cloud (Microsoft 2020c). Migration of the data into the relational format is done with a custom made python tool. Currently, the only supported platform for Power BI gateway is Windows, so it needs to be hosted on a separate virtual machine from the rest of the software components.

The initial MongoDB database which gets updated after each completed Jenkins build is in a Linux Virtual Machine instance. The virtual machine has plenty of resources to support containerized applications and each software component in this product will be containerized to reduce combability issues and support portability in case the project needs to be moved to another instance. These environments are provided by Nokia Networks.

Microsoft Power BI and Python are selected for the project due to their prevalence at Nokia. Most developers at Nokia know how to develop new functionality with Python and

non-critical software components are developed using it. Nokia uses Microsoft products in their work, and their Azure offering and support for Microsoft products is comprehensive, therefore Power BI is used for data analytics.

6.2 Data's current format

Data is stored into a MongoDB database where it needs to be fetched using MongoDB's supported queries. Attachment 2 represents the data model of the stored data from a Jenkins built in the MongoDB database. Data needs to be represented in relational format for Microsoft Power BI. Currently, Microsoft or MongoDB's developer Atlas do not provide drivers to on-premises MongoDB instances, which would be needed to provide real-time data streaming (Microsoft 13.3.2020).

Translation to relational form could be achieved using mongosqld (MongoDB 2020). However, it uses 'drdl'-format to define the relational model, which is not well documented and needs to be connected to Power BI using ODBC-drivers and queries need to be translated to MongoDB queries each time, which may result in bad performance compared to direct access to a SQL database. Also, MongoDB aggregation functions of 'drdl'-format do not support extracting values out of embedded documents, if they are not in their separate collection or embedded in a list.

6.2.1 Problems with the JSON-model

Current format of the JSON-model in which the documents are stored in the MongoDB database is slightly problematic. Key "testcases" can contain zero to *n* unique testcase objects as values, where the *n* is number of testcases. In relation modeling, this is referred as One-to-Many relationship: each *one* Jenkins build contains *n (many)* number of testcases. The related documents are embedded into the parent document as objects, which may cause long documents.

As stated in chapter 3.1, non-relational databases do not need to follow any structural rules, except for ECMA-404 'The JSON Data Interchange Syntax standard' and this model is as the standard defines. However, MongoDB organization provides a guide to model one to many relationships (MongoDB 2020c). In the MongoDB's documentation is stated: "A potential problem with the embedded document pattern is that it can lead to large documents, especially if the embedded field is unbounded.". Here this problem is encountered: if *n* or number of testcases is even 50, program needs to load the full, rather large document into computer memory at once. It is not a problem when handling few documents, but when handling tens of thousands, it will impact the program's performance. It could be

considered as best practice to embed related unique values as a list or even as a separate collection, performance wise (MongoDB 2020c). As stated in chapter 6.2, this model blocks us using the mongosqld's 'drdl'-format to translate the documents into relational models, as each embedded document would produce its own unique table and no relations can be formed. Thus, migration to SQL database is required.

6.3 Modeling an SQL database for migration

I selected PostgreSQL for the migration database as they have an official Docker image and it is a fully featured database system. Attachment 3 represents the relational data model in normalized database diagram format modeled after the JSON-model represented in the attachment 2. Each node represents a table in the database. Tables are connected to each other with lines which indicate relationships between tables: one-to-many and many to one are the only relationships present here. In the tables are listed the column names and datatypes. PK and FK indicate Primary Key and Foreign Key constraints, which respectively mean the unique identifiers in the database table and unique identifier of a related table.

The database is in fourth normal form (Kent 1982). The error table could be normalized further as the same error could occur more than once, but also the same error could occur multiple times with minor changes, which would lead into inconsistencies.

The modeled database is stored in PostgreSQL server, which is hosted inside a container on PostgreSQL's official Docker image. Ports from the container are opened so that the container is accessible from the internal network.

6.4 Migration software

Migration software is the part of the data pipeline that translates the JSON documents into relational format. The software is built using Python 3.7 combined with a few libraries. The libraries included are SQLAlchemy, psutil, psycopg2, PyMongo and pytz. These dependencies are managed using pypip. Code follows PEP8 style guide and automatic styling was used when developing with PyCharm or Visual Studio Code.

Most values can be set using command line parameters when running the software. These include the database addresses and migration modes: full migration or only sync. The full migration can be skipped, if the software is taken down for maintenance. Source code is confidential to the customer, but some details are given with code snippets.

6.4.1 Source Control Management

The software's source code is managed using internal GitLab servers. Each commit is committed using as descriptive commit messages as possible and merged into the repository. Documentation for the software is also stored inside the git repository.

6.4.2 Migration and syncing process

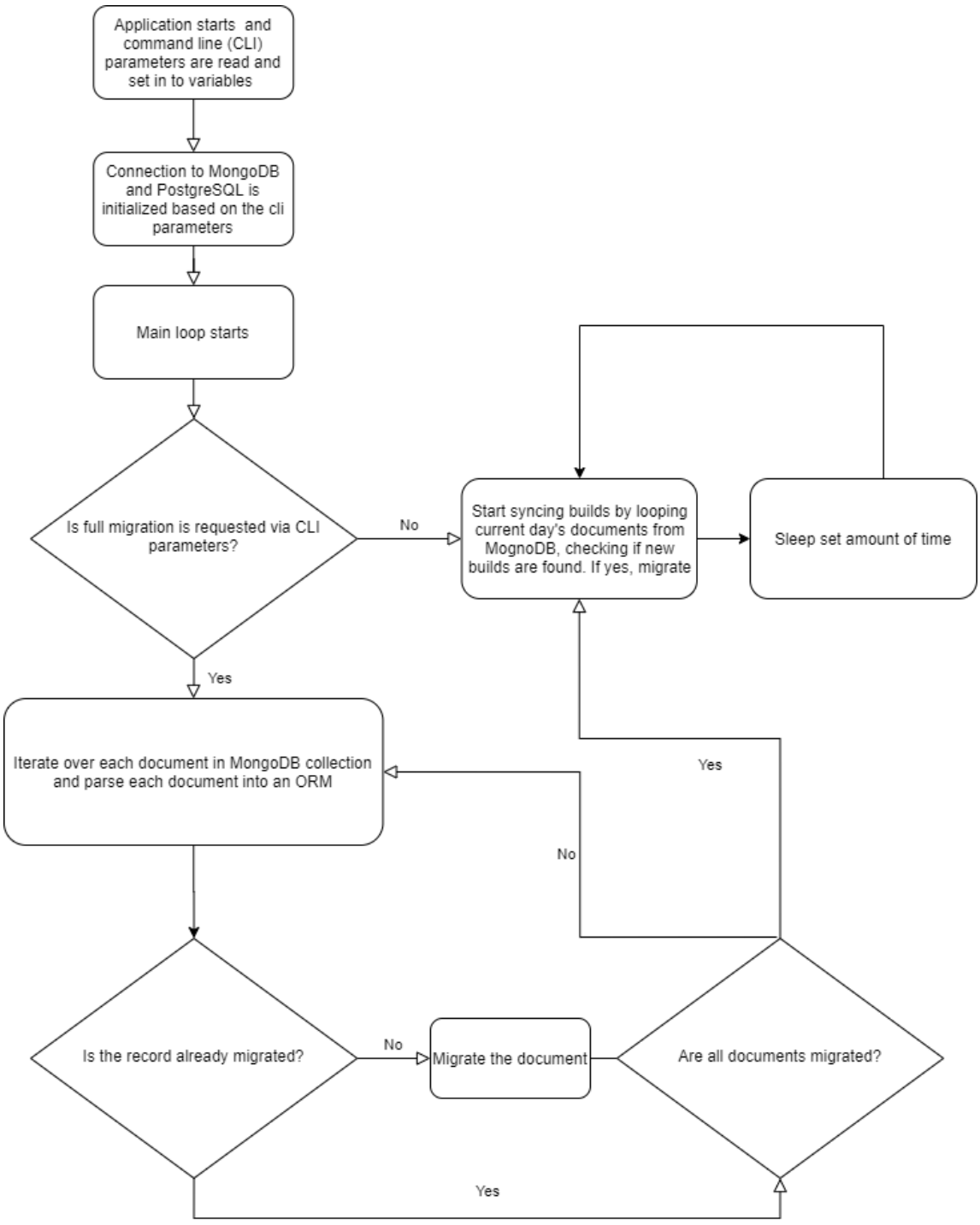


Figure 8 Flowchart of the simplified migration process

SQLAlchemy is a widely used library, that makes it possible to model a relational database using ORM (SQLAlchemy). It helps significantly with managing relational data inside a program. Objects can be modeled as classes inside a program, then the SQLAlchemy takes care of translating any transactions to an SQL-type transaction. The programmer does not necessarily need write any SQL queries, although custom queries can be created using the library.

SQLAlchemy is used in the project to handle database transactions and store the data in objects inside the program. In figure 9 is defined one ORM model. The '__init__' function can be called to initialize a new object using the model. Each variable represents a column and datatype is given as a parameter. Relationships are set with the "relationship()" function. Each table in the database diagram have their own ORM class. SQLAlchemy has a function "create_all()" to build a database using the defined ORM classes.

```
class Testcase(Base):
    __tablename__ = 'Testcase'

    id = Column(Integer, primary_key=True, nullable=False)
    name = Column(String, nullable=False)
    doc = Column(String, nullable=False)

    tags = relationship("TestcaseTag", backref='Testcase')

    def __init__(self, name, critical, doc):
        self.name = name
        self.doc = doc
```

Figure 9 An ORM class from 'Testcase' table using SQLAlchemy.

The documents are fetched using the open source PyMongo library (MongoDB 2020e). It allows efficient and secure querying of data from MongoDB databases and handling the returned documents as Python dictionaries.

```
cursor = mongo_db.find({})
for doc in cursor:
    print(doc)
```

Figure 10 Example on listing all documents in a MongoDB collection using PyMongo. Variable mongo_db includes the initiated connection to a MongoDB collection.

After the document is fetched, it is mapped into an ORM, then the program checks, if record with the data can already be found from the database. Each related table goes through the same process. In case any problems occur during the migration, each query and parsing process is contained in a try-catch function. The program tries again a set

number of times, if the error occurs due to connection issues and then skips the document. Each error is reported to the logging module.

```
if testcase_in_db is None:
  testcase_in_db = Testcase(name=test['name'], critical=critical, doc=test['doc'])
  sql_db.add(testcase_in_db)
```

Figure 11 Forming an object using the previously defined ORM

After migration is concluded, the program shifts into syncing mode. The current days documents are checked through similarly as in full migration and after each completed iteration the program sleeps for a set amount of time. This implementation is not optimal, but the used database was an older version and did not support data streaming.

6.4.3 Logging

All the events inside the program are logged using Python's included logging module. It allows logging events on multiple levels: info, warning, error, critical, log and exception. In my implementation, each successful and failed migration's information is logged for further inspection. The log's formatting is set using the logging module and includes the date, time and logging level.

```
logging.info(msg="RobotResult with _id " + obj_id + " succesfully migrated")
```

Figure 12 Example of logging migrated document

6.4.4 Environment

The software is designed to run in a Docker container. The container is defined in Dockerfile format so it can be deployed anywhere. Container is then connected to the same container network as the PostgreSQL container, so they can communicate together.

```
FROM python:3
MAINTAINER "Joni Taajamo"
COPY ./migrator/ /home/
WORKDIR /home/
RUN pip install -r requirements.txt
CMD [ "python", "./app.py" ]
```

Figure 13 Example Dockerfile for the program

6.4.5 Documentation

An installation guide, architecture documentation and developer guide are provided as internal documentation. Each of these are typed in markdown format and stored into the git

repository. The program is self-documented as well as possible with descriptive variable and function naming. Supportive comments are added to the code, if needed.

6.5 Power BI report

Finally, when the data is migrated to the PostgreSQL database, it can be fetched for visualization with Power BI. The report was initially developed on Power BI Desktop platform and then published to the Power BI service. Data from the PostgreSQL is transferred to the Power BI service using Microsoft provided Power BI gateway and by setting up an automatically updating dataset from the Power BI service. Currently, new data can be updated every 30 minutes to the service using the gateway.

Attachment 4 represents the ‘testcase’ view in the Power BI report. User can view single testcases execution history, all testcase executions for a time period or all testcases for a Jenkins build. Multiple filters can be applied at the same time to filter out irrelevant data.

Users have rights to edit the report and they can easily add new charts and tables into the report using Power BI.

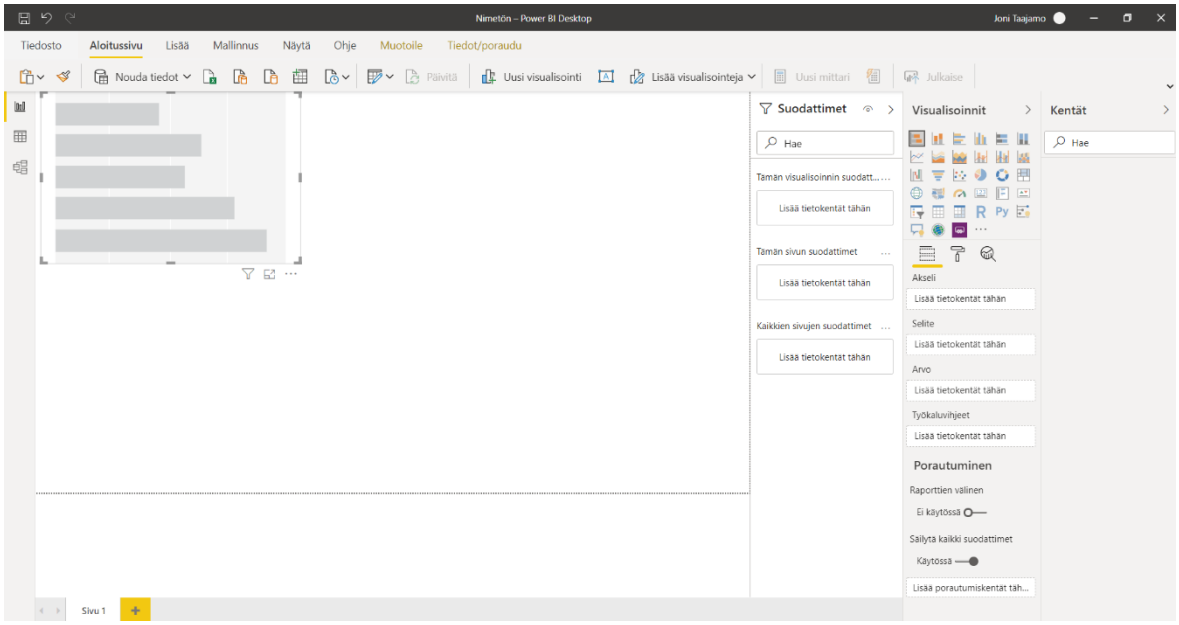


Figure 14 Power BI Desktop report editing mode

7 Results

The final product of the thesis shows data analytics from the related data as planned. From my point of view, the results were satisfactory taking the technologies used into account.

7.1 Technical quality

The software quality was satisfactory from a technical standpoint. Considering the amount of resources available for the project, the software performs reliably and is maintainable. More testing for the migration software would be needed, if further development is done as current testing is limited. A higher level of abstraction for the software would also result in better quality software and higher maintainability.

Microsoft Power BI was the biggest bottleneck of the project. It has restrictions in handling and updating large amounts of data, which causes slowdowns at times. Nonetheless, its ease of use goes a long way to compensating for this weakness. The product is still under active development by Microsoft so I'm sure the whole software will improve greatly. If I were to design the architecture of the projects system again, I would certainly investigate the usage of Power BI API introduced during the project. The API could improve the performance of the Power BI report.

7.2 User satisfaction measurement

User satisfaction of the system is measured in using End User Computer Satisfaction (EUCS) cross-tabulation developed by Doll and Torkzadeh and later re-introduced for BI measuring by Dedić and Stanier (2017). Dedić and Stanier propose doing the measurement in 4 phases. To keep measurement brief, I combined phases 3 and 4 introduced by Dedić and Stanier. In table 1, questions for the measurement are introduced. Questions are answered using levels 1-5. For example, in question "Is the system accurate?" the range represents the accuracy of the system: 1 is not accurate, 3 is somewhat accurate and 5 is very accurate.

	Questions
Content	1. Does the system provide the precise information you need?
	2. Does the information content meet your needs?
Accuracy	3. Is the system accurate?
	4. Are you satisfied with the accuracy of the system?
Format	5. Do you think the output is presented in a useful format?
	6. Is the information clear?
Ease of use	7. Is the system user friendly?
	8. Is the system easy to use?
Timeliness	9. Do you get the information you need in time?
	10. Does the system provide up-to-date information?

Table 1. Cross-tabulation of EUCS phases 3 and 4 combined. (Dedić & Stanier. 2017)

7.3 User satisfaction results

The questions were sent and answers were received via email to the participants. Three users reported their experience by answering the EUCS questions presented in chapter 7.2. The participants for the inquiry were members of the user team: scrum master of the user team, technical leader of the team and the manager of the team. The results of the EUCS questions are represented in table 2.

Question	Tech. Lead.	Scrum M.	Manager
1	5	4	5
2	5	4	5
3	5	5	5
4	5	5	5
5	4	4	4
6	4	5	4
7	4	4	4
8	5	4	5
9	5	4	5
10	5	5	5

Table 2. EUCS results.

The average answer to all questions is 4,6 out of 5 and the mean is 5 out of 5. The results can be interpreted as high user satisfactory for the product. Feedback from the users outside the inquiry has also been generally positive.

Users also reported that one of the main goals were reached: manual labor wasn't required to form reports that were done with Excel prior the project. The project also inspired other similar projects in other teams, which was also one of the goals of the project.

8 Conclusion

The development process was finished within the time limits set. At the end of the project, the pipeline and tooling are in use and are in a state where they can be developed further if needed.

8.1 Scheduling and usage of Scrum

Initially the project started slowly: it was difficult to define what the project was focused on and what technologies should be used for it. When the subject was decided, the work itself progressed well.

As introduced in chapter 5, the software was developed incrementally using Scrum. The process was natural: the software's functionality improved after each sprint and something of value was represented to stakeholders. Then during the sprints, the software was improved and at least one working feature was introduced at the end of each sprint. The project lasted total of 5 2-week sprints.

The state of the software after each sprint was as follows:

- report was available through desktop version of the Power BI for one of the tested product types.
- the same report was available through web interface with few more visualizations
- all the tested product types were available
- filtering functionality for the result
- "drillthrough" (Power BI feature) functionality for the report (final state)

Most requirements were fulfilled during the development phase. New requirements arose at the beginning of the project and some were refined with the team's feedback.

However, some requirements could not be satisfied within the project's schedule as they needed changes to other systems. The missing functionality will be added to the project later, when the missing systems are available for use. The project may be developed further inside the team, but I will be moving to another team at Nokia. I will still provide support for the product in case it is needed.

8.2 Technology choices

I don't have profound experience from any other BI system to compare Power BI to its competitors, but it seems like Power BI is rising in popularity currently. Based on this project, I could suggest using Power BI in other projects like this, but it is also worth investigating the offerings from competitors.

In migration applications, a more powerful language than Python could be used, but Python has its advantages, like ease of learning. Each engineer needs to evaluate their own use cases before selecting technologies. In some other data heavy application, choosing a language with better performance and optimizing the data handling algorithms would be needed. For this project Python was a good choice because of the fast implementation and no need for optimal performance.

8.3 Learning process

During the development of the project, I learnt not only about the technologies that were used, but also about managing and structuring my work. Sometimes I had to make one functionality work at a time and then move to the next, because the previous functionality was dependent on the other. The project also forced me to learn about handling of complex architecture and designing each part of the system as their separate smaller system. This design approach helps to reduce the complexity of the whole system as each part can be updated solely but was also required for this project as the data needed to be handled separate of the representation.

After each sprint, I held a short demo session just as planned. I had not demoed any software alone to an audience before this project. Keeping the demo session short and clear was not always easy, but I think I improved in demoing after each sprint.

The most difficult part of the development process was getting the correct licenses for the software used in the project. Licenses were required for Microsoft Power BI and Windows Virtual Machine. They were provided, but their delay slowed down the work at some points meaning that I needed to focus other aspects of the project when waiting. The need for licenses should always be sorted out at the earliest stage possible in any project. This project reminded me of that aspect of developing software in professional environment.

Sources

Aijaz, A. Release Frequency: A Need for Speed. 2019. DevOps Zone. Accessible: https://dzone.com/articles/release-frequency-a-need-for-speed. Accessed: 9.2.2020.

CircleCI 10.2.2020. CircleCI pricing. Accessible: https://circleci.com/pricing/. Accessed: 10.2.2020.

Codd, E. F. 1970. A relational model of data for large shared data banks. Communications of the ACM, 13, p. 377-387.

Dedić, N. & Stanier, C. 2017. Measuring the Success of Changes to Existing Business Intelligence Solutions to Improve Business Intelligence Reporting. HAL. France.

Ecma international. 2017. Standard ECMA-404, The JSON Data Interchange Syntax, 2nd edition. Switzerland.

Fowler, M. 5.1.2006. Continuous Integration. Accessible: https://martinfowler.com/articles/continuousIntegration.html. Accessed: 9.2.2020.

Google for Education. Python Introduction. Accessible: https://developers.google.com/edu/python/introduction. Accessed: 22.3.2020.

Grenning, J. 2002. Planning Poker or How to avoid analysis paralysis while release planning.

Haerder, T. & Reuter A. 1983. Principles of transaction-oriented database recovery. ACM Computing Surveys. 15, 4, p. 287-317.

Hibernate 10.2.2020. What is Object/Relational Mapping. Accessible: http://hibernate.org/orm/what-is-an-orm/. Accessed: 10.2.2020.

Humble, J. & Farley, D. 7.9.2010. Continuous Delivery: Anatomy of the Deployment Pipeline. Informit. Accessible: https://www.informit.com/articles/article.aspx?p=1621865&seqNum=2. Accessed: 24.3.2020.

IBM. 15.5.2019. Containerization. Accessible: https://www.ibm.com/cloud/learn/containerization. Accessed: 27.3.2020.

Jenkins 9.2.2020. Jenkins User Documentation. Accessible: https://jenkins.io/doc/. Accessed: 9.2.2020.

Jenkins 7.12.2017. Terminology. Accessible: https://wiki.jenkins.io/display/JENKINS/Terminology. Accessed: 9.2.2020.

Kent, W. 1989. A Simple Guide to Five Normal Forms in Relational Database Theory. IEEE Computer Society Press. p. 120-125.

Kim, G.; Humble, J.; Debois, P. & Willis, J. 2016. The DevOps Handbook: How to crea- world-class agility, reliability, & security in technology organizations. IT Revolution Press. Portland, OR, United States.

Laine, H. History of SQL. Accessible: https://www.cs.helsinki.fi/u/laine/tuelip/sql_material/sql_history.html. Accessed: 10.2.2020.

Lambert, T. 2017. Why is python slower than C? Quora. Accessible: https://www.quora.com/Why-is-python-slower-than-C/answer/Terry-Lambert. Accessed: 29.3.2020.

Marazzi, A. 20.6.2016. Building a Data Pipeline from Scratch. Accessible: https://medium.com/the-data-experience/building-a-data-pipeline-from-scratch-32b712cfb1db. Accessed: 26.3.2020.

Microsoft. 13.3.2020a. Data sources in Power BI Desktop. Accessible: https://docs.microsoft.com/en-us/power-bi/desktop-data-sources. Accessed: 20.3.2020.

Microsoft. 2020b. Power BI. Accessible: https://powerbi.microsoft.com/en-us/. Accessed: 15.3.2020.

Microsoft 2020c. Power BI gateway. Accessible: https://powerbi.microsoft.com/en-us/gateway/. Accessed: 20.3.2020.

MongoDB 2020a. MongoDB. Accessible: https://www.mongodb.com/. Accessed: 10.2.2020.

MongoDB. 2020b. mongosqld. Accessible: https://docs.mongodb.com/bi-connector/master/reference/mongosqld/. Accessed: 26.3.2020.

MongoDB. 2020c. Model One-to-Many Relationships with Embedded Documents. Accessible: https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/. Accessed: 23.3.2020.

MongoDB. 2020d. Introduction to MongoDB. Accessible: https://docs.mongodb.com/manual/introduction/. Accessed: 23.3.2020.

MongoDB. 2020e. PyMongo 3.9.0 Documentation. Accessible: https://api.mongodb.com/python/current/. Accessed: 29.3.2020.

Morris, J. Practical Data Migration. 2006. BCS Learning & Development Ltd. United Kindom.

Nokia Oyj 9.2.2020. What we do. Accessible: https://www.nokia.com/about-us/what-we-do/. Accessed: 9.2.2020.

PostgreSQL. 2020. psql. Accessible: https://www.postgresql.org/docs/9.2/app-psql.html. Accessed: 28.3.2020.

Python. Applications for Python. Accessible: https://www.python.org/about/apps/. Accessed: 29.3.2020.

Robot Framework 10.2.2020. Robot Framework. Accessible: https://robotframework.org/#documentation. Accessed: 10.2.2020.

Royce, W. Managing the Development of Large Software Systems. 1970. Proceedings of IEEE WESCON. 26, p. 328-338.

SQLAlchemy. SQLAlchemy. Accessible: https://www.sqlalchemy.org/. Accessed: 29.3.2020.

Scrum 10.2.2020. What is scrum. Accessible: https://www.scrum.org/resources/what-is-scrum. Accessed: 10.2.2020.

Spear, M.E. 1952. Charting Statistics. OCLC 166502. New York.

The Computer Language Benchmarks Game. Accessible: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html. Accessed: 29.3.2020.

Tozzi, C. 31.5.2020. The Limitations of NoSQL Database Storage: Why NoSQL's Not Perfect. ChannelFutures. Accessible: https://www.channelfutures.com/cloud-2/the-limitations-of-nosql-database-storage-why-nosqls-not-perfect. Accessed: 10.2.2020

Travis CI 10.2.2020. Travis CI plans. Accessible: https://travis-ci.com/plans. Accessed: 10.2.2020.

Wu, C.F.J. 1997. Statistics = Data Science?.

Attachments

Attachment 1. Product backlog before first sprint.

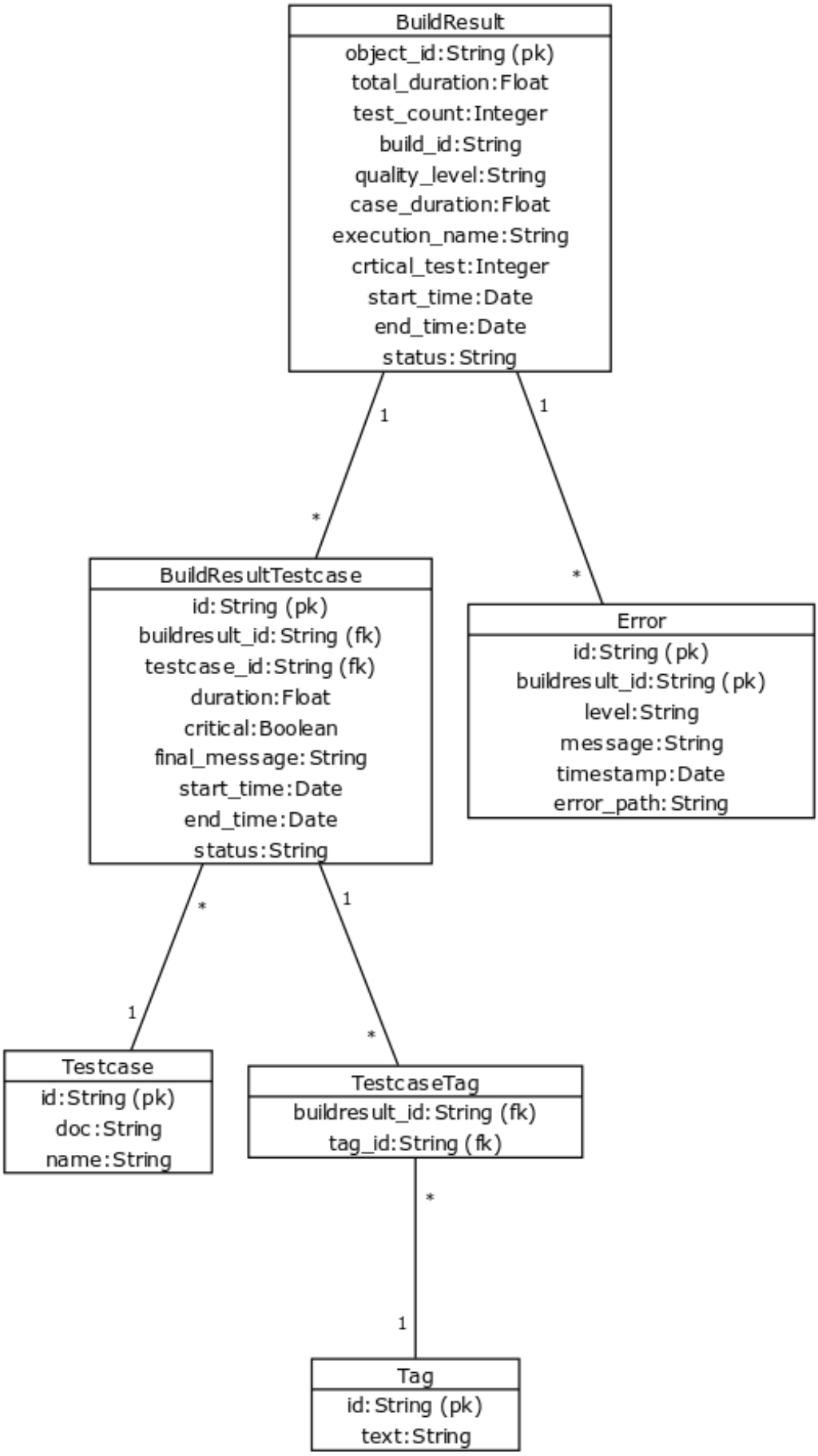
Priority	User story	Status	Story Points	In Sprint	Acceptance criteria
A	As a tester, I can view all builds as a list in PBI Desktop from one collection	In progress	6	Sprint 1	
A	As tester, I can view any visualization about real data in PBI Desktop	In progress	4	Sprint 1	Data migration is built and running on a server platform
A	As a tester, I can view any visualization in browser	Accepted by Customer	4	Sprint 2	Power BI service license required
A	As a tester, I can view one test case's execution history with statuses.	Accepted by Customer	6	Sprint 2	View of which build failed and error code why it failed.
A	As a tester, I can view if system_1 tickets has been created for this test case.	Accepted by Customer	6	Sprint 3	
A	As a tester, I can view if system_2 tickets has been created for this test case.	Accepted by Customer	5	Sprint 3	
B	As a tester, I can view if there are changes in that domain test case repository or software repository.	Accepted by Customer	5	Sprint 4	Link to diff file from analysis.
C	As a tester, I can view domain test case success history, which test case failed and which build.	Accepted by Customer	6	Sprint 4	
C	As a domain developer, I can view list of top 10 longest test case execution time	Accepted by Customer	3	Sprint 4	
C	As a tester, I can view all test case results of build visualized in a compilation view	Accepted by Customer	4	Sprint 4	

Attachment 2. Mock JSON data. Some key value pairs removed due to confidential data.


```
{
    "_id" : ObjectId("111111111"),
    "totalDuration" : 1111,
    "All Tests" : "2/0",
    "testcases" : {
        "testcase_1" : {
            "duration" : 0.21212122333,
            "critical" : "yes",
            "finalMessage" : "Success",
            "name" : "testcase_1",
            "doc" : "testcase_1",
            "startTime" : "20191204 08:49:41.634",
            "endTime" : "20181204 08:49:41.994",
            "tags" : [
                "tag1",
                                    "tag2",
                                    "tag3"
            ],
            "status" : "PASS"
        },
        "testcase_2" : {
            "owner" : "email@email.com",
            "duration" : 0.21212122333,
            "critical" : "yes",
            "finalMessage" : "Success",
            "name" : "testcase_1",
            "doc" : "testcase_1",
            "startTime" : "20191204 08:49:41.634",
            "endTime" : "20181204 08:49:41.994",
            "tags" : [
                    "tag1",
                    "tag2",
                    "tag3"
            ],
            "status" : "PASS"
        }
```

```
    },
    "buildID" : "123456789",
    "qualityLevel" : "QL1",
    "caseDuration" : 21.1231412,
    "executionName" : "jenkins_job1",
    "Critical Tests" : "2/0",
    "startTime" : "20191204 08:47:56.954",
    "endTime" : "20191204 08:51:43.616",
    "errors" : {
        "WARN" : {
            "level" : "WARN",
            "message" : "Warning!",
            "timestamp" : "20191204 08:49:57.931",
            "errorPath" : "path/to/error"
        }
    },
    "status" : "PASS",
}
```

Attachment 3. Relational model based on attachment 2 JSON model.



Attachment 4. Single testcase’s execution history. Confidential elements covered.