Universiteit Leiden
The Netherlands

ICT in business and the public sector

Defining and measuring the maintainability of Splunk apps

Marlo Brochard

Supervisors:
Prof.dr.ir. J.M.W. Visser & Dr. C.J. Stettina MSc

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

20/08/2021

Abstract

Introduction: Although the use of big data analytic platforms such as Splunk has increased, the way in which the software quality of these products can be measured has not been well established. In contrast to other programming languages such as Python and Java, the mindset of Splunk developers is not focused on the maintainability of their apps or software quality in general. It is often focused on creating new apps on top of the Splunk platform to provide more valuable information with high performance. However, because the use of the Splunk platform has increased and many organisations rely on the insights gained from numerous apps deployed on the Splunk platform for making important decisions, it is important to change the mindset and also include the focus on software quality. According to ISO/IEC 25010 [iso], functional suitability is a characteristic of software quality which verifies the correctness of the data provided by a system or product. Therefore, poor software quality could, for example, lead to a financial loss when the data is unreliable because the apps are often used by businesses to make important decisions. Furthermore, maintainability is also one of the characteristics of software quality according to ISO/IEC 25010 on which we will focus during this master thesis. The maintainability costs of a software program is a high percentage of the overall cost of developing that program which would be reduced when developing high maintainable software. Furthermore, according to Meso and Jain [MJ06], organizations value systems and products that can be better adapted and improved to serve new business needs or to compensate for changes in the underlying systems which can be achieved by ensuring higher maintainability [MD06].

Objectives: In this thesis, the main objective was to provide a definition of maintainability, which is an aspect within software quality defined by ISO/IEC 25010, for Splunk apps and create automated scripts for measuring and assigning a maintainability score to Splunk apps. In other words, we want to provide an instrument to organisations that develop and /or rely on Splunk apps to control and manage their maintainability.

Methods: In order to achieve our main objective, we first created an understanding of the current situation regarding software quality and Splunk apps by conducting a theoretical investigation. This will result in an initial quality model by conducting literature research in combination with the knowledge and experience of the thesis author as a Splunk developer. Second, we have validated the initial quality model by conducting two focus groups in combination with a survey. After that, we have finalized the quality model by creating a selection of quality metrics that can be used to measure the maintainability of Splunk apps derived from either the survey and focus group answers, literature and/or existing quality models. We then converted the obtained quality model into a design of the Splunk app that can be used to continuously measure and visualize the maintainability score of a Splunk app that is being developed. Finally, we have evaluated the design of the tool and the model that it embodies by conducting an empirical investigation at SMT, a medium-size company that offers data-driven solutions by using the SMT Enriched Data Analytics Platform.

Results: The validation shows that the Splunk experts are satisfied with the quality model and the design of the tool that implements it as it matches the assessment criteria of experts. Furthermore, they indicated that the quality model and tool are useful in practise.

Conclusion: The quality model and created design are suitable for providing developers continuous insights into the maintainability of their Splunk app and thereby improving their development process and making better informed decisions based on valuable data. As mentioned before, organizations value systems and products that can be better adapted and improved to serve new business needs or to compensate for changes in the underlying systems which can be achieved by ensuring higher maintainability. By using the quality model

and created design of the tool, the developers can use the provided information to improve the maintainability of their Splunk app and thereby develop more adaptable/changeable Splunk apps. Furthermore, the development of high maintainable code would reduce the maintainability costs because less time have to be spent on the maintainability activities. It should also ensure that Splunk apps with a higher software quality are published and therefore provide more reliable Splunk apps that can be used by businesses to make important decisions. However, this is based on the validation by conducting two focus groups, a survey and three final interviews in which I presented the quality model and created design. Therefore, it is still needed to create the tool and to validate the model by using the tool and to perform a real-world validation of the maintainability ratings which is also discussed in this thesis.

Contents

Acknowledgement

I would like to thank my supervisor Prof.dr.ir. J.M.W. Visser for his advice, feedback and support during my master thesis. I was not able to complete my master thesis, as it is now, without him as my supervisor.

I would also like to thank my second supervisor Dr. C.J. Stettina MSc for being my second supervisor and for the feedback given at the end of my thesis. And lastly, I would like to thank SMT, in particular Aniel, for the collaboration and support during my master thesis.

1 Introduction

Nowadays, big data becomes more and more important as more data comes available which can be used to create better insights of the current situation. The main difference between the traditional term "data" and "big data" can be explained by looking at five key characteristics. First of all, the volume of big data is much higher than ever before. The forecast of the volume of big data in 2020 was indicated with 59 zettabytes by Statista [Hol21]. The velocity is the second characteristic which is also much higher than the traditional data. Big data is available at real-time or near real-time which gives companies the ability to act quickly. The variety of data is the third characteristic we will discuss. Nowadays, we see a transition of businesses moving their activities to an online platform which means that more and more types of data becomes available [rev14]. Furthermore, the data should be of the right accuracy and stored, processed and analysed at reasonable costs which is related to the veracity and value characteristic of big data respectively [Maz16].

The power of successfully using big data becomes increasingly important. A company that is able to measure based on big data is also able to manage more precisely in comparison with companies not using or not successfully using big data. The management can use these measurements to make better predictions, and more important smarter decisions, because it is based on facts instead of gut feeling [rev14]. In order to successfully use big data, monitoring tools and data platforms have been created to transform data into knowledge. By using these monitoring tools and data platforms, big data can be preprocessed and visualized in order to create a better understanding of the current and previous situation of the company.

1.1 Software quality

It is becoming increasingly important to have a good software quality because more and more products and services depend on these software related products [AMM19]. In research from Arcos-Medina and Mauricio [AMM19], they indicate that there are two types of software quality: product quality and software development process quality [AMM19]. Here, the product quality consists of eight characteristics: functionality, efficiency, compatibility, usability, reliability, security, maintainability, and portability which are further subdivided into 31 sub-characteristics [iso]. And the software development process is a complex process that influences the end results. It consists of several activities such as requirements engineering and reviewing.

Maintainability is one of the characteristics of product quality according to ISO/IEC 25010 [iso]. This characteristic is further divided into the following subcategories:

1. Modularity: how large is the impact of making changes to one component on other components?
2. Reusability: how easy or difficult can a component be used in other systems?
3. Analysability: how easy or difficult is it to diagnose which components of a software should be modified and what the impact is on the software product?
4. Modifiability: how easy or difficult can the software product be changed without causing defects or a lower software quality?

5. Testability: how easy or difficult can the software product be tested on several test criteria after a modification?

There is already a lot of research done on the maintainability aspect of software quality and the importance of a good maintainable software in the software development domain [ASC02][CALO94b]. According to [CALO94b], the maintainability of software products has become a very important requirement within the software development domain. In the book by Fred Brooks, he claimed that the maintainability costs of a widely used software program is approximately 40 percent or more of the total cost of developing that program [BJ95] or even 45 to 60 percent claimed by parikh [PZS83]. Furthermore, according to Meso and Jain [MJ06], organizations value systems and products that can be better adapted and improved to serve new business needs or to compensate for changes in the underlying systems which can be achieved by ensuring higher maintainability [MD06].

In general, maintainability measures the performance of the maintenance activities by the technical staff of an organisation. However, during this master thesis we want to focus on measuring maintainability by directly observe a system's source code instead of indirectly by looking at the maintenance activities. There are already a few quality metrics proposed as indicators for the maintainability characteristic of a software product such as the Maintainability Index [CALO94a] and the SIG maintainability model [HKV07].

1.2 Splunk

One of the big data platforms available on the market is called Splunk [spla] which is a powerful tool for searching and exploring all the big data that is available and relevant for a company. At the time of writing this master thesis, Splunk has around 6.000 employees and 91 active customers. By using Splunk, they want to remove the barriers between data and using the data to make important decisions. The Splunk platform can be divided into three categories which are IT operations, business intelligence and security. According to the book written by Carasso [Car14], Splunk is mainly used by system administrators, network administrators, and security gurus but can be used by many other audiences.

Splunk developers are able to create Splunk apps that are built on top of the Splunk platform or to extend the platform, for example, by making a connection between Splunk and another platform such as TOPDesk [mcg]. These apps can only be used within a Splunk Enterpise deployment or Splunk Cloud [splb]. In most cases, a Splunk app consists of one or more dashboards that make a useful visualization of the data. Furthermore, it consists of saved configurations and knowledge objects such as lookups, alerts, saved searches, data inputs and many more [splb]. A more advanced Splunk app could also consist of, for example, custom Python scripts or Javascript. Here, it is important to note that there are a lot of restrictions from Splunk for writing Python code that is compatible with Splunk. In this master thesis, we will focus on Splunk apps containing dashboards, saved configurations and knowledge objects within Splunk and custom Python scripts.

Because big data and the tools to analyse big data become more important, the importance of a good quality of those tools increases. Therefore, measuring the maintainability, which is one of the eight characteristics of software quality, of Splunk apps is also becoming more important.

However, there is not much research on this specific topic. As mentioned before, Splunk is one of the larges big data monitoring platforms available on the market. Within the big data monitoring domain, there are many more alternative platforms such as Elastic [ela], Dynatrace [dyn] and Datadog [Dat16].

1.3 Existing tools

In the area of software quality related to Splunk there are a few tools available on the market to measure the code within a Splunk app. Examples of such a tool are AppInspect [app], PyTest [pyt] and a more Python specific tool called PyLint [Log].

Splunk AppInspect is a tool that validates a splunk app by using a number of criteria defined by Splunk which should help the development team in building a high quality and robustness app. By validating a splunk app, AppInspect will generate a report that gives the development team feedback about the missing criteria. It is also required to pass all the criteria validated by AppInspect before a Splunk app can be published on Splunkbase, which is the marketplace for Splunk apps. The domains on which Splunk AppInspect validates a Splunk app are [app]:

- Structure
- Feature set
- Security
- Adherence to the guidelines for Splunk Cloud apps

But Splunk AppInspect does not validate the maintainability of a Splunk app.

PyTest is a tool or framework that can be used to write test scripts for Python code which assure better software. Splunk has developed a PyTest Splunk addon which is an open-source dynamic test plugin especially for splunk apps. It gives Splunk developers the opportunity to test knowledge objects, CIM compatibility and index time properties [pyt]. So the main goal of the PyTest Splunk addon is to validate whether the Splunk app works as it should work. But it does not measure whether a piece of code is easy to maintain.

PyLint is a tool that is created to validate Python code on programming errors, helps enforcing a coding standard and sniffs for code smells [Log]. Pylint will generate a report that gives the development team feedback about how they can improve the Python code. It also generates feedback about the code's complexity such as code duplication. This tool is created for Python and therefore not directly linked to the Splunk language. But as mentioned before, we will focus on Splunk apps containing dashboards, saved configurations and knowledge objects within Splunk and custom Python scripts. Therefore, we could use PyLint for our maintainability measurements on the custom Python scripts because it is an open-source software which makes it possible to extend PyLint with our own checks.

1.4 Research Questions

Because of the aforementioned information, it is important for Splunk app developers to have a tool available that measures and monitors the maintainability of their app. But as mentioned in Section 1.3, there is no Splunk tool available that can be used to measure and monitor the maintainability of Splunk apps. This means that we have to find out which components of a Splunk app are subjected to software quality, which quality metrics can be used to measure the maintainability of these components and how these quality metrics can be combined into a single quality model. The information mentioned above led me to the following research question:

How can we continuously measure and monitor the maintainability of Splunk apps?

This main research question can be divided into the following sub-research questions:
Research question 1: Which components of a Splunk app are subjected to maintainability?

Research question 2a: How can we define good maintainable code for each component defined in RQ1?
Research question 2b: Which quality metrics are suitable for measuring the aspects defined in RQ2a?

Research question 3a: How can we define an empirically-based rating system for each quality metric from RQ2?
Research question 3b: How can these quality metrics from RQ2 be combined into a single quality model?

Research question 4a: Is the quality model from RQ3 experienced as useful in practice by Splunk related companies?
Research question 4b: What is the effect of the newly created tool on the strategic goals of a company?

Splunk app developers can use this tool and the model that it embodies to get more insights in the maintainability of their Splunk app and thereby improve the software quality.

1.5 Approach

In order to answer the research questions mentioned in the aforementioned section, we will perform the following steps:

1. Create an understanding of the current situation regarding maintainability and Splunk apps by conducting a theoretical investigation. After that, we will develop an initial quality model by conducting literature research in combination with the knowledge and experience of the thesis author as a Splunk developer.

2. We have to validate the initial quality model by conducting two focus groups in combination with a survey.

3. After that, we will finalize the quality model by creating a selection of quality metrics that can be used to measure the maintainability of Splunk apps derived from either the survey and focus group answers, literature and/or existing quality models. Furthermore, we will create a benchmark rating system for each metric defined in our pre-final quality model.

4. Create a tool by converting the obtained quality model into a design of the Splunk app that can be used to continuously measure and visualize the maintainability score of a Splunk app that is being developed.

5. Evaluating the created tool/quality model by conducting an empirical investigation at SMT which has been done by conducting three interviews.

These steps will be explained in more detail in Section 5.

1.6 Thesis overview

This chapter contains the introduction of my master thesis. In Section 2, we will give a brief explanation of the fundamental concepts used in this master thesis. In Section 3, we will explain the identified problem and the contribution of this master thesis. In Section 4, we will discuss papers that are related to our research field. In Section 5, we will discuss our approach and thereby explain how we are going to answer the research questions. In Section 6, we will present our initial quality model which will be validated by conducting two focus groups and a survey. In Section 7, we will briefly explain our focus groups and survey questions and execution. And we will also make an analysis of the answers obtained from the focus groups and survey questions. In Section 8, we will make a selection of the quality metrics and combine them into a single quality model. In Section 9 and 11, we will transform the quality model into a design of the Splunk app and then validate the created tool. In Section 10, we will define thresholds and a benchmark rating system for each metric in our final quality model. Lastly, in Section 12 and 13 we will discuss our results and draw a conclusion based on the obtained results. We will also mention some interesting ideas for further research. This master thesis will be conducted in collaboration with SMT [Jva20] and supervised by Prof.dr.ir. J.M.W. Visser [joo].

2 Fundamental concepts

During this section, we will give a theoretical explanation of Splunk, software quality, quality metrics and how to develop a benchmark rating system.

2.1 Splunk

In this chapter, we will give a brief introduction of the Splunk language and it's concepts in order to provide enough information needed to understand the rest of the paper. As mentioned in the introduction, Splunk is one of the big data platforms available on the market [spla] which is a powerful tool for searching and exploring all the big data that is available and relevant for a company. At the time of writing this master thesis, Splunk has around 6.000 employees and 91 active customers. By using Splunk, they want to remove the barriers between data and using the data to make important decisions. The Splunk platform can be divided into three domains which are IT operations, business intelligence and security. According to the book written by Carasso [Car14], Splunk is mainly used by system administrators, network administrators, and security gurus but can be used by many other audiences.

Splunk developers are able to create Splunk apps that are built on top of the Splunk platform or to extend the platform, for example, by making a connection between Splunk and another platform such as TOPDesk [mcg]. These apps can only be used within a Splunk Enterpise deployment or Splunk Cloud [splb]. As mentioned before, we will focus on Splunk apps containing dashboards, saved configurations and knowledge objects within Splunk and custom python scripts. Now, we will explain the different components of such a Splunk app in order to provide enough information to understand these concepts when they are discussed in coming chapters.

2.1.1 SPL searches

As mentioned in the first section of this chapter, Splunk is a platform that can be used to store and use data in order to get better insights of the data. In order to extract information from the large amount of data stored within Splunk, the Search Processing Language has been developed by Splunk. The Search Processing Language can be used to write searches to extract the desired data from all the data stored within Splunk. Its syntax originates from a combination of the Unix pipeline and SQL language optimized for time series data which scope includes data searching, filtering, modification, manipulation, insertion, and deletion [splc]. The basic structure of a Splunk query is as follows:

Figure 1: The basic structure of a Splunk query [Lue17].

First of all, a search and filter should be defined indicating which data should be used in the search. In Figure 2, we search for all the data that has been identified with the source type "access∗". Here, a source type identifies the data structure of a data point and indicates how Splunk should format this data during the indexing process. After specifying which data should be used in the

search, it is possible to modify that data which is also called mungeing. In Figure 2, we modify the data by creating a new field called "KB" by dividing the existing field "bytes" by 1024 using the eval command. In the Search Processing Language there are more than 140 different search commands that can be used to extract the desired information from the data [Lue17]. In order to get an overview of all the possible search commands that can be used in the Search Processing Language, I would refer to the documentation published by Splunk [sea]. After applying the necessary modifications on the data, it is possible to use the search commands for creating the desired report of the data. In Figure 2, we use the "stats" command to calculate and return the total number of KB's and distinct client IP-addresses. Last but not least, it is possible to clean the output of the report in order to make it more reader friendly. In Figure 2, we used the "rename" command to rename the output fields such that it is easier to read and understand.

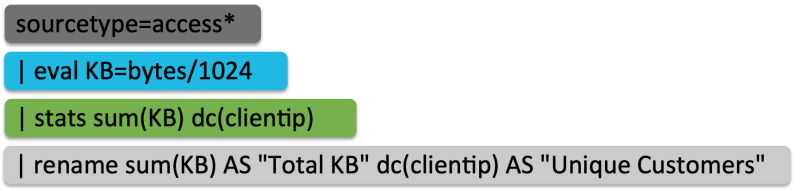


Figure 2: An example of a Splunk query [Lue17].

2.1.2 Dashboards

A dashboard in Splunk is used to make an understandable visualization of the data that has a certain business meaning. A dashboard consists of two or more panels, consisting of a SPL search, which are different kinds of visualizations of the data and therefore provides different kinds of information about the data. A dashboard panel can take different types which are single values, gauges, maps, event lists, tables, charts and time-charts. The main difference between a chart and a time-chart is that a time-chart visualizes the data over time and a chart is a snapshot of the current time [Inc]. A Splunk dashboard also consists of one or more dashboard filters such as a time window, type of context and many more custom filters that can be created by the developers. In Figure 3, we highlighted most of the components of a Splunk dashboards in order to get a better idea of these concepts.

A dashboard makes use of simple XML source code to define the content shown on a dashboard and it's behaviour. The simple XML source code can be edited either via the Splunk web interface or in your own developing environment. It consists of different components such as a fieldset, rows, panels and more. Within the fieldset label, there should be one or more input types defined which corresponds to the dashboard filters in Figure 3. After that, the beginning of a row should be defined where a Splunk dashboard can consists of multiple rows. Within a row, mulitple panels can be defined which are visualized next to each other as can be seen in Figure 3. In each panel, it should be defined which visualization type is used, which search runs behind the panel and it is possible to set panel options and drilldowns. A drilldown could be a link to a different dashboard, a search, URL or it could cause a panel to be visible when clicking on the given panel. The main structure of the simple XML source code is visualized in Figure 4.

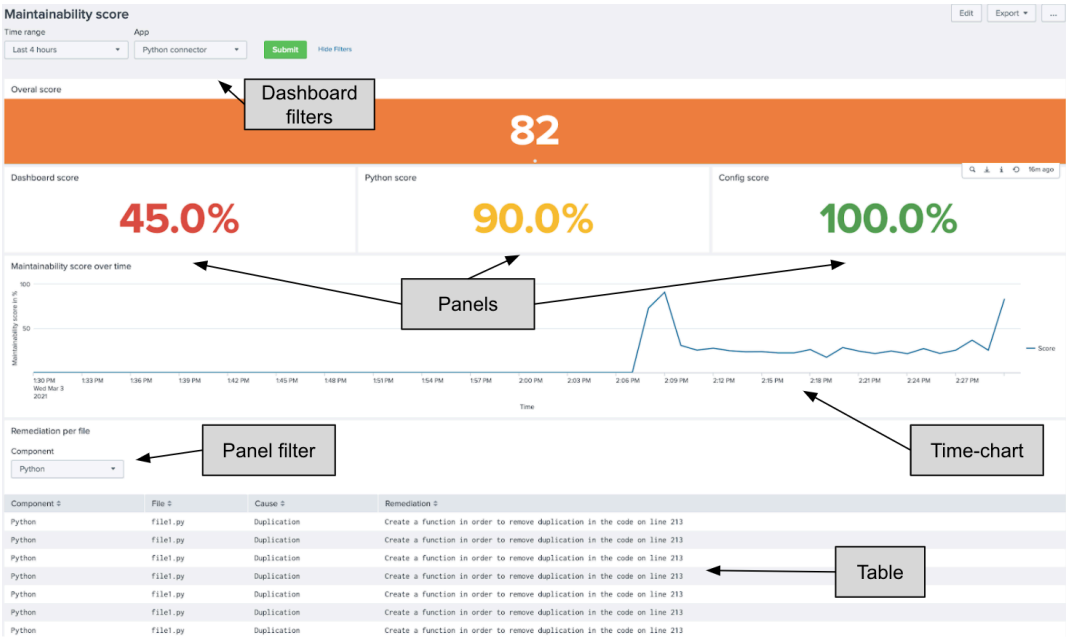


Figure 3: An example of a dashboard in Splunk. Here, the dashboard filters and some panel types are highlighted.

2.1.3 Config files

Besides dashboards and SPL searches, a Splunk app also consists of several configuration files, also known as `.conf` files, which contains all configuration information for a specific app. Within a Splunk app there are 2 different types of configuration files which are default and local configuration files. The configuration files located in the `default` folder of a Splunk app are preconfigured files which should not be edited by the user of the app. The configuration files located in the `local` folder of a Splunk app can be used to edit the default configurations without editing the preconfigured configuration files located in the `default` folder. In the Splunk language there are more than 56 different configuration files that can be used to configure a Splunk app. In order to get an overview of all the possible configuration files that can be used in the Splunk language, I would refer to the documentation published by Splunk [con]. In these configuration files, it is only possible to use settings that are defined by Splunk for each type of configuration file. This should be taken into account during this master thesis, because these settings could be used by each Splunk app and have always the same meaning. It is also possible to create your own custom configuration files that can, for example, be used by your custom Python files.

2.1.4 Knowledge objects

A Splunk app usually contains one or more knowlegde objects that can be used by the users of the app. A knowledge object is a user-defined entity that can be used to enrich the existing data in Splunk or to extract specific information about the data in Splunk. Knowledge objects can take many different types such as lookups, alerts, saved searches, data inputs and many more [splb]. According to the documentation published by Splunk, the knowledge objects could be divided into five categories which are [kno]:

```
<form> <!--Start of a dashboard-->
  <fieldset><!--Start of the dashboard filters-->
    <input> <!--One or more dahsboard filters-->
    </input> <!--End of one dashboard filter-->
  </fieldset>
  <row> <!--Start of a dashboard row-->
    <panel> <!--Start of a dashboard panel on this row-->
      <title></title>
        <single> <!--Type of dashboard panel-->
        <search> <!--Start of the search behind the panel-->
          <query> <!--The search query-->
          </query>
        </search>
        <option></option> <!--Options for the dashboard panel-->
        <drilldown></drilldown> <!--Drilldowns on click-->
      </single>
    </panel>
  </row> <!--End of a dashboard row-->
</form> <!--End of a dashboard-->
```

Figure 4: An example of the simple XML source code of a dashboard in Splunk. Here, the different components are highlighted.

1. Data interpretation
2. Data classification
3. Data enrichment
4. Data normalization
5. Data models

Within a Splunk app usually a lot of knowledge objects exists varying from alerts to lookups which should be maintained.

2.1.5 Custom Python files

In most cases, a Splunk app consists of one or more dashboards that makes a useful visualization of the data. Furthermore, it consists of saved configurations and knowledge objects such as lookups, alerts, saved searches, data inputs and many more [splb]. A more advanced Splunk app could also consist of, for example, custom Python scripts or Javascript. Here, it is important to note that there are a lot of restrictions from Splunk for writing Python code such that it is compatible with Splunk. In this master thesis, we will focus on Splunk apps containing dashboards, saved configurations and knowledge objects within Splunk in combination with custom python scripts. For a more detailed explanation of the Splunk language and its possibilities, I would refer to the documentation published by Splunk.

2.2 Software quality

Nowadays, software becomes more and more important and is part of the revenue generation process in almost all industrial domains [Rom02]. This causes that the software quality of these software

related products becomes more important as more people and businesses rely on these software products. A poor software quality could, for example, lead to financial loss, mission failure, injuries or even worse losing of human lives. The cause of, for example, a financial loss by having a poor software quality is especially the case when looking at Splunk apps because they are often used by businesses to make important decisions. Instead of using their gut feelings, they use these Splunk apps which create useful visualizations or generate reports using the data collected by the company in order to make certain decisions.

Besides a higher importance of software quality also the time-to-market for new software is of high importance which creates a challenge between the requirements of a short time-to-market and producing high quality software [Rom02]. This could be related to the Iron triangle which has become very important in measuring the success of a project [Atk99]. The Iron triangle consists of three main criteria which are time, costs and quality shown in Figure 5. Here, the balance between time, costs and quality is one of the main factors for success or failure in the software development domain. According to the book by Diomidis Spinellis, the quality factor of the Iron triangle is the most difficult factor as it cannot be changed immediately by the actions of the management and because solving poor software quality is very difficult [Spi06]. This makes it important for the

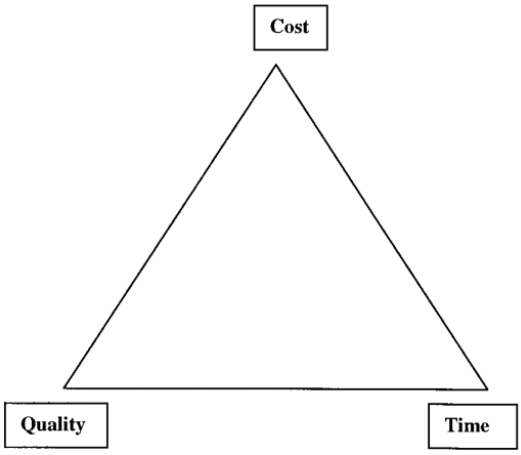


Figure 5: The Iron triangle consisting of the three criteria time, costs and quality [Atk99].

development team to have a faster way of monitoring and measuring the software quality of their software ensuring little time have to be spend on software quality.

In research from Arcos-Medina and Mauricio [AMM19], they indicate that there are two types of software quality: Product quality and software development process quality [AMM19]. Here, the product quality consists of eight characteristics: functionality, efficiency, compatibility, usability, reliability, security, maintainability, and portability which are further subdivided into 31 sub-characteristics [iso] shown in Figure 6. And the software development process is a complex process that influences the end results. It consists of several activities such as requirements engineering and reviewing. During this master thesis, we will focus on the product quality type of software quality and therefore not taking into account the development process quality type of software quality.

2.2.1 Maintainability

Maintainability is one of the characteristics of product quality according to ISO/IEC 25010 [iso]. The definition of maintainability used during this master thesis is the definition provided by the IEEE Standard Glossary of Software Engineering Terminology [Rad90] formulated as follows:

> *The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment* [Rad90].

This is a well-established and widely used [DJ03][HF02][Lan02] definition in the software development domain. The maintainability characteristic is further divided into the following subcategories:

1. Modularity: how large is the impact of making changes to one component on other components?
2. Reusability: how easy or difficult can a component be used in other systems?
3. Analysability: how easy or difficult is it to diagnose which components of a software should be modified and what the impact is on the software product?
4. Modifiability: how easy or difficult can the software product be changed without causing defects or a lower software quality?
5. Testability: how easy or difficult can the software product be tested on several test criteria after a modification?

In general, maintainability measures the performance of the maintenance activities by the technical staff of an organisation. However, during this master thesis we want to focus on measuring maintainability by directly observe a system's source code instead of indirectly by looking at the maintenance activities. There are already a few quality metrics proposed as indicators for the maintainability characteristic of a software product such as the Maintainability Index [CALO94a] and the SIG maintainability model [HKV07]. However, there is not yet a widely used model to rate the maintainability of a software system [Spi06]. During this master thesis, we will focus

Figure 6: ISO/IEC 25010: The product quality model [iso].

on the maintainability aspect of software quality which is divided into three subcategories by using a combination of the Boehm's Quality Model [BBL76] and the ISO standards [iso] which are understandability, modifiability and testability. Here, we will focus on the maintainability aspect

of software quality and then in particular of Splunk apps. In this master thesis, maintainability should give answers to the questions: how easy is it to understand, modify and retest the software product? In order to measure the maintainability of Splunk apps, we will define several metrics that can be measured and combined into a single quality model which will be further explained in the next sub-chapter.

2.3 Metrics

The history of software metrics goes back to the mid-1960's when the first software metric was used which is the so-called lines of code metric. This metric was used to measure the programmers productivity, programming effort and the quality of the software by measuring, for example, the LOC per programmer per month or the number of defects per thousand lines of code [FN99][FN00]. However, the first dedicated book about software metrics was published on 1976 [Gil76]. According to N.E. Fenton and M. Neil [FN99], the first prediction model for predicting software quality using metrics was published on 1971 by Akiyama where they used the KLOC metric to make a prediction about the complexity of a software system. In the mid-1970s, the use of only the LOC metric for measuring the productivity of programmers or the complexity of code was criticised. At that moment, they came to the conclusion that there was an urgent need for more diverse metrics which where programming language independent resulting in an increased interest in software metrics [FN99].

The term software metrics has been clearly described by Fenton and M. Neil [FN99] as:

> *In fact software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterise properties of software code through to models that help predict software resource requirements and software quality. The subject also includes the quantitative aspects of quality control and assurance - and this covers activities like recording and monitoring defects during development and testing.*
> ([FN99], p.360)

2.3.1 Product metrics

According to the book by Kan [Kan03], software metrics can be divided into three categories which are product metrics, process metrics and project metrics. Product metrics are used to describe the characteristics of a software such as the volume, complexity and performance of the software. The process and project metrics are used to describe the development process and project characteristics respectively. In this master thesis, we are focused on measuring the maintainability score of a Splunk app by using source code characteristics such as volume and complexity which is a subcategory of product metrics [Kan03]. In this field, a lot of research has already been done on defining metrics for measuring maintainability by using source code characteristics. Examples of such metrics defined by other researchers are focused on complexity by McCabe [McC76], volume by Halstead [Hal77] and cohesion by Emerson [Eme84].

In this master thesis, we will develop a quality model consisting of quality metrics that can be used to measure the maintainability aspect of Splunk apps by using source code characteristics.

The metrics used in the presented quality model will be explained in Chapter 6 and combined into a single quality model in Chapter 8.

2.4 Benchmark rating system

Software metrics have been around since mid-1960's when the first software metric was used which is the so-called lines of code metric. This metric was used to measure the programmers productivity, programming effort and the quality of the software by measuring, for example, the LOC per programmer per month or the number of defects per thousand lines of code [FN99][FN00]. However, the use of software metrics in order to support the decision-making process have generally not been successful [FN00]. In order to increase the use of metrics to support the decision-making process, it is important to define meaningful threshold values that are not only based on expert opinions or a few observations [AYV10]. In this master thesis, we will also define meaningful threshold values for the product metrics defined in our quality model.

2.4.1 Thresholds

It is important to define meaningful threshold values for each product metric in order to increase the use of metrics to support the decision-making process. Instead of having the metric value based on expert opinions or a few observations, it should be based on empirical results from the measurement data of a benchmark dataset [AYV10]. In research from Alves, Ypma and Visser [AYV10], they have developed a method to derive threshold values empirically by using a benchmark dataset, which is in our case a dataset containing code of multiple open-access Splunk apps. This method ensures that it:

> *[...] (i) bring out the metric's variability between systems and (ii) help focus on a reasonable percentage of the source code volume* [AYV10]

This method, based on a benchmark dataset, consists of six successive steps and will be explained using the McCabe metric [AYV10]:

1. **Metric extraction**: for each Splunk app, *System*, in our benchmark dataset and for each Python method, *Entity*, we calculate the metric value, *Metric*, and weight value, *Weight*. Here, the weight value is defined as the number of lines of that entity. Using the McCabe metric, this could result in a metric value of, for example, 17 and a weight value of 120.

2. **Weight ratio calculation**: for each entity, we will calculate its percentage within the system by dividing the number of lines of that entity by the total number of lines, the sum of all weights, in that system. Using the same example, this could result in a weight value of, for example, 120 of a given entity and a total number of lines in that system of 120.000 and thus representing 0.01% of the overall system.

3. **Entity aggregation**: for each system, we aggregate the weight values for each metric value. Using the same example, this could result in all entities of a given system with a McCabe value of 17 represent 1.2% of the overall system. And all entities of the same system with a McCabe value of 10 represent 10% of the overall system.

4. **System aggregation**: for the benchmark dataset, we aggregate the weights (of previous step) for all systems. Here, it is important to normalize the weights for the number of systems in the benchmark dataset such that the sum of the weights remains 100%. Using the same example, this could result in a McCabe value of 17 representing 0.08% of the overall code in the benchmark dataset. And a McCabe value of 10 represent 9% of the overall code.

5. **Weight ratio aggregation**: in order to define meaningful thresholds, we have to calculate the maximal metric value per percentage of the overall code in our benchmark dataset. Using the same example, this could result in 60% of the overall code has a maximal McCabe value of 2.

6. **Threshold derivation**: in order to derive the threshold values, we have to choose the percentage of the overall code we want to represent. For example, we want to represent 70%, 80% and 90% of the overall code for deriving the threshold values for the McCabe metric. This could result in the threshold values of 6, 8 and 14 respectively meaning that a McCabe value of 14 represent the most risk and a value of below 6 the lowest risk knowing that a higher McCabe value is worse [McC76].

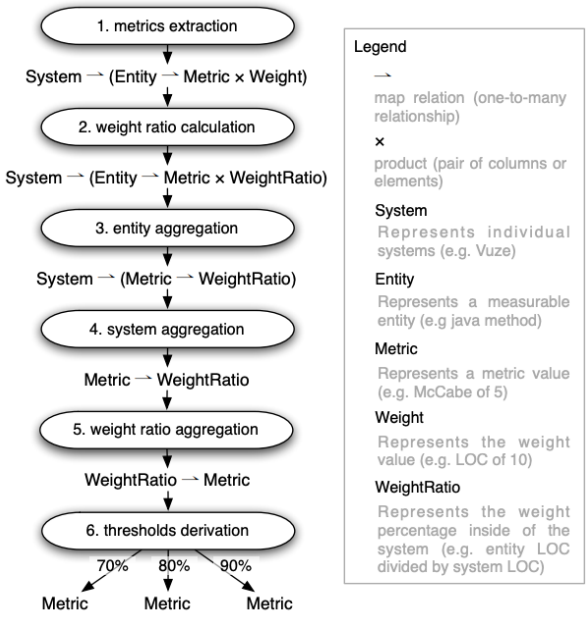


Figure 7: An overview of the method used to derive the threshold values using a benchmark dataset proposed by Alves, Ypma and Visser [AYV10].

After defining the threshold values, we will use these values to define risk profiles for each metric using the benchmark dataset. This will be explained in more detail in the next subsection.

2.4.2 Rating system

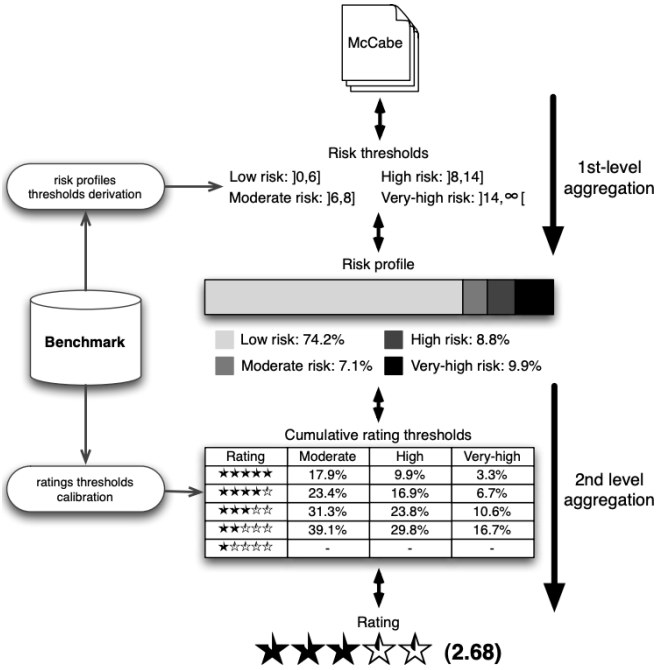
In order to rate a system based on a metric value, we have to define a rating system for each metric. In order to derive these rating systems, we will use the approach proposed by Alves, Correia and

Visser [ACV11] using a benchmark dataset. The approach consists of two different processes which are called the 1st-level aggregation and the 2nd-level aggregation. The 1st-level aggregation will aggregate the measurements into a risk profile by using the metric thresholds explained in the previous chapter. A risk profile consists of different risk levels which depends on the number of metric thresholds and indicates the percentage of the overall code that belongs to each risk level. The 2nd-level aggregation will aggregate the risk profile into a rating by using the cumulative rating thresholds. We will explain this in more detail by using an example with the McCabe metric of system X.

In the 1st-level aggregation, we will create a risk profile for system X. The McCabe metric thresholds are 6, 8 and 14 meaning that a McCabe value of 14 or higher represent the most risk and a value of below 6 the lowest risk knowing that a higher McCabe value is worse [McC76]. Using these thresholds, we will create a risk profile for system X by calculating the percentage of the overall code that belongs to a McCabe value of below 6, between 6 and 8, between 8 and 14, and 14 or higher. This is calculated by dividing the total number of lines linked to each McCabe value, which was the weight value in the previous chapter, with the total number of lines in system X resulting in the relative size of system x that falls in each risk category. For example, there are 95,262 lines linked to a McCabe value of below 6 and the system contains 128,316 lines which will result that 74.2% of the system is in the low risk category. This could result in, for example, a risk profile containing 74.2% of code in the Low risk, 7.1% for Moderate risk, 8.8% for High risk, and 9.9% for Very-high risk which is also shown in Figure 8 [ACV11].

In the 2nd-level aggregation, we will create a rating for system X. First of all, we will calculate the cumulative relative size of each risk category using the risk profile. The cumulative relative size is calculated by adding up the relative size of each risk category plus all higher categories. For example, the cumulative moderate risk of system X is calculated by 7.1%, which is the relative size of moderate risk, plus 8.8%, which is the relative size of high risk, plus 9.9%, which is the relative size of very high risk, which results in 25.8%. This is calculated for system X for each category, except for the lowest category because this will always be 100%, resulting in 25.8% for moderate risk, 18.7% for high risk and 9.9% for very high risk. These values are used to determine the rating by comparing them with the cumulative rating thresholds of the 5-stars rating system for the McCabe metric. For example, the rating thresholds for a 4-star rating could be 23.4% moderate risk, 16.9% high risk and 6.7% very high risk. And the rating thresholds for a 5-star rating could be 17.9% moderate risk, 9.9% high risk and 3.3% very high risk. This means that a system containing a cumulative moderate risk of between 23.4% and 17.9%, a cumulative high risk of between 16.9% and 9.9%, and a cumulative very high risk of between 6.7% and 3.3% will get a 4-star rating. According to Figure 8, system X would get a 3 star rating when we compare the relative size of each category with the rating thresholds [ACV11] because this is the highest rating where we do not exceed a set of cumulative rating thresholds.

In order to derive a 5-stars rating system, we have to define the cumulative rating thresholds for each category and star rating. These thresholds are calculated by using the algorithm proposed in a paper by Alves, Correia and Visser [ACV11]. First of all, we have to create a risk profile for each system in the benchmark dataset. Then these risk profiles are aggregated into a rating system by defining the minimum thresholds that can be used to divide the systems of our benchmark



Rating	Moderate	High	Very-high
★★★★★	17.9%	9.9%	3.3%
★★★★☆	23.4%	16.9%	6.7%
★★★☆☆	31.3%	23.8%	10.6%
★★☆☆☆	39.1%	29.8%	16.7%
★☆☆☆☆	-	-	-

Figure 8: An overview of the method used to derive a rating for a system based on a metric value using a benchmark dataset proposed by Alves, Correia and Visser [ACV11].

dataset into 5 categories. The input parameters for the calibration algorithm are the cumulative risk profiles of each system in our benchmark dataset and a partition indicating the number of systems that should be assigned to each rating. Here, the risk profiles for each system in our benchmark dataset are calculated as explained in the above paragraphs. The partition depends on the desired distribution of the systems per rating. For example, when an uniform distribution is desired, the partition will be 20% of the systems in our benchmark dataset for each rating when creating a 5-stars rating system. This means that when having a 100 systems in our benchmark dataset, the partition will be 20-20-20-20-20%. In order to explain the algorithm, we will use the following example:

- Our benchmark dataset contains 100 systems.
- We want to create a 5-stars rating system.
- We have 4 risk categories low, moderate, high and very high risk but the low risk category is not taken into account by the algorithm because its value is always 100% for each system as explained before.
- We want an uniform distribution of the systems in our benchmark dataset.

The general steps of the calibration algorithm are as follows [ACV11]:

1. First of all, an ordered matrix will be created of which the columns represent the risk categories which are moderated, high and very high risk. The rows of the matrix represent the values of the cumulative risk profile of each system resulting in 100 values in each column because our benchmark dataset contains 100 systems and thus 100 risk profiles are created.

2. After creating the matrix, the algorithm will walk through all positions in order to find the optimal thresholds. For each rating, in our example rating 1 to 5, the algorithm first tries to find the initial set of thresholds such that it is conform the partition value. For the first rating, this is done by taking the first value of each category in the ordered matrix which is the lowest percentage. Then the algorithm will keep going to the next value of each risk category till the thresholds ensure a distribution of the systems in our benchmark dataset conform the partition which is in our example 20 systems.

3. After finding the initial thresholds, the algorithm will try to find the optimal thresholds by processing each risk category individual instead of collective. In order to find the initial thresholds, the algorithm goes to the next value for each risk category after one iteration. In the optimization part, the algorithm will try to lower the thresholds by going to the previous value of the first risk category and not for all risk categories. This means that in the first iteration it will go to the previous value, compared to the value found in the initial part, of the moderated risk category and checks whether the distribution is still conform the partition. This is done till it is not conform the partition anymore and saves the threshold for the moderate risk and goes to the next risk category.

4. The initial and optimization part of the algorithm will be executed for each rating, which is in our example 5. At the end of the algorithm, it returns the optimal thresholds for each risk category and rating such that it is conform the desired distribution which is in our example 20 systems per rating. An example of such an output is shown in Figure 8.

For more details about the calibration algorithm and the pseudocode, I would refer to the paper by Alves, Correia and Visser [ACV11] in which more details about this approach is explained.

3 Problem statement

As mentioned in Section 1, big data and the tools to analyse big data becomes more important, and therefore also the importance of a good software quality of those tools increases. Software quality can be explained by looking at eight characteristics which are: functionality, efficiency, compatibility, usability, reliability, security, maintainability, and portability which are further subdivided into 31 sub-characteristics [iso]. This master thesis has the goal to create a quality model containing code level product metrics, which can help the Splunk app developers with measuring the maintainability score of their Splunk app. By answering the main research question, mentioned in Section 1, we can provide the Splunk app developers with information about the maintainability score of their Splunk app by translating the quality model into a tool which will be a Splunk app. This Splunk app should contain the functionality and dashboards to measure and visualize the maintainability score. These dashboards show the overall maintainability score which is broken down into the maintainability score of each component of a Splunk app which makes it easier to detect the main issue causing a low maintainability. It is important to get an indication of the current maintainability score because a low maintainability score means a low performance of the maintenance activities by the technical staff of an organisation which lead to higher costs and effort. Furthermore, according to Meso and Jain [MJ06], organizations value systems and products that can be better adapted and improved to serve new business needs or to compensate for changes in the underlying systems which can be achieved by ensuring higher maintainability [MD06].

Nowadays, there is much research on software quality, more specifically on maintainability, related to the Python language which is only one component of a Splunk app. But there is not much research on maintainability related to the big data platform called Splunk. This causes that Splunk apps are difficult to understand, modify and use by Splunk users who are not familiar with that Splunk app because there is not much research and no helpful tool on how to develop a high maintainable Splunk app. This problem was encountered by the development team and I during the development of several Splunk apps which is one of the reasons for doing this research project. The second reason for doing this project is the growing importance of developing high quality software as more businesses rely on these software products, mentioned in Section 2.2. Because there is not much research done on this specific subject, we have to obtain answers to the questions: which components of a Splunk app are subjected to software quality? Which quality metrics can be used to measure maintainability of Splunk apps? Which data do we need in order to measure these metrics? How can we ensure that our target audience wants to adopt our developed quality model?

This paper makes the following academic contribution to the research on measuring the maintainability of Splunk apps in the field of computer science:

1. A theoretical and empirical investigation into the best practises of creating a Splunk app, the standards for each component of a Splunk app and the quality metrics that can be used to measure the software quality of Splunk apps.

2. A developed quality model that can be used to measure and monitor the software quality of Splunk apps.

3. A brief discussion about how the obtained results can be generalized to data monitoring

platforms in general.

4. An investigation into the effects of the newly created tool on the strategic goals of a company.

Furthermore, this master thesis provides an instrument to organisations that develop and /or rely on Splunk apps to control and manage their maintainability.

4 Related Work

In this chapter we will discuss studies that are relevant for this research project. Measuring code level product metrics to say something about the maintainability of a software product is a subject that has already been investigated. First of all, we will discuss several existing quality models such as McCall and SIG. Then, we will explain the different approaches for creating a quality model such as the Goal-question-metric approach.

4.1 Software quality models

4.1.1 McCall

The quality model developed by McCall et al. [MRW77] indicates that the software quality attributes can be divided into three levels which are software, factors and metrics. First of all, a system is considered as:

> *the programs and documentation associated with and resulting from the software development process.*
> ([MRW77], p.14)

which could be divided into three product activities. These activities could be described as (1)Product Operation, which refers to the quality of the operations such as the efficiency of the operations and the ability to provide the desired information by the user; (2) Product revision, which refers to error correction and system adaptation; (3)Product transition, which refers to the ability to be moved between different environments [OPR03]. The quality model considered the quality characteristic maintainability to be part of the product revision activity which definition is defined as follows:

> *The effort required to locate and fix an error in an operational program*
> ([MRW77], p.20)

The quality model indicates that the quality metrics for measuring maintainability are linked to five criterion which are consistency, simplicity, conciseness, modularity and self descriptiveness. Here, the quality metrics should provide an indication of the quality of the end products. According to Ortega, Pérez and Rojas [OPR03], this is also one of the major contributions of the McCall quality model where it defines the relationship between the quality characteristics and metrics. An overview of the quality model is shown in Figure 9.

4.1.2 Boehm

The quality model of Boehm et al. [BBL76] is similar to the quality model developed by McCall et al. [MRW77], as it also present the quality model as a hierarchical model consisting of four main levels. The highest level is defined as the General Utility of the software product which means that the usefulness of the model is the most important characteristic. If a software product is not useful, it is a waste of time, effort and money to develop that particular software. The general utility is divided into three conditions which are maintainability, As-is Utility and portability. According to Boehm et al. [BBL76], the definition of maintainability is defined as:

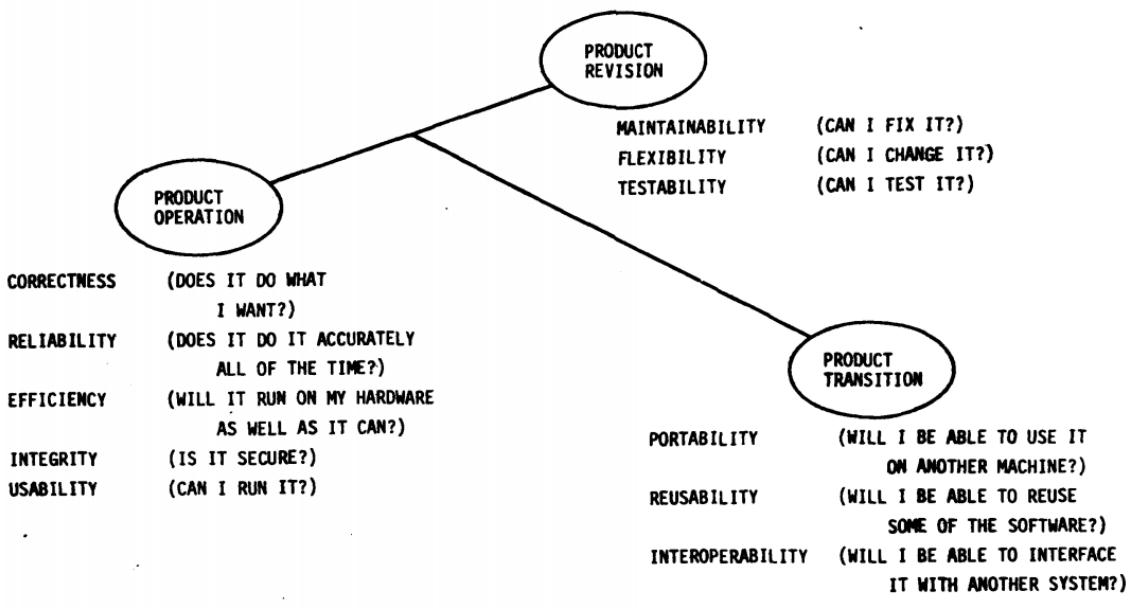


Figure 9: An overview of the quality model proposed by McCall et al. [MRW77].

> *How easy is it to maintain (understand, modify, and retest)?*
> ([BBL76], p.4)

Here, maintainability is further sub-divided into understandability, modifiability and testability. These sub-characteristics are further linked to nine primitive characteristics: consistency, accountability, accessibility, communicativeness, self-descriptiveness, structuredness, conciseness, legibility and augmentability [BBL76]. The nine primitive characteristics can be used to define quantitative metrics that gives an indication of the higher level characteristics. One of the major differences between the quality model developed by McCall et al. is that the quality model by Boehm et al. also adds the hardware yield characteristics [OPR03]. An overview of the quality model is shown in Figure 10.

4.1.3 ISO/IEC 25010

The International Standards Organization (ISO) published during the years 2001 to 2004 an international consensus on the terminology for quality characteristics of software products which consists of one International Standard (IS) and three Technical Reports (TR) [HKV07]. Since the year 2011, ISO-9126 is replaced by ISO-25010 which is a quality model that can be used as a product quality evaluation system. This quality model describes which quality characteristics should be taken into account when evaluating the quality of a software product [iso]. Here, the quality model consists of eight characteristics: Functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability which are further subdivided into 31 sub-characteristics [iso]. An overview of the quality model is shown in Figure 11.

Maintainability is one of the characteristics of product quality according to ISO/IEC 25010 [iso]. This characteristic is further divided into the following subcategories:

1. Modularity: how large is the impact of making changes to one component on other components?

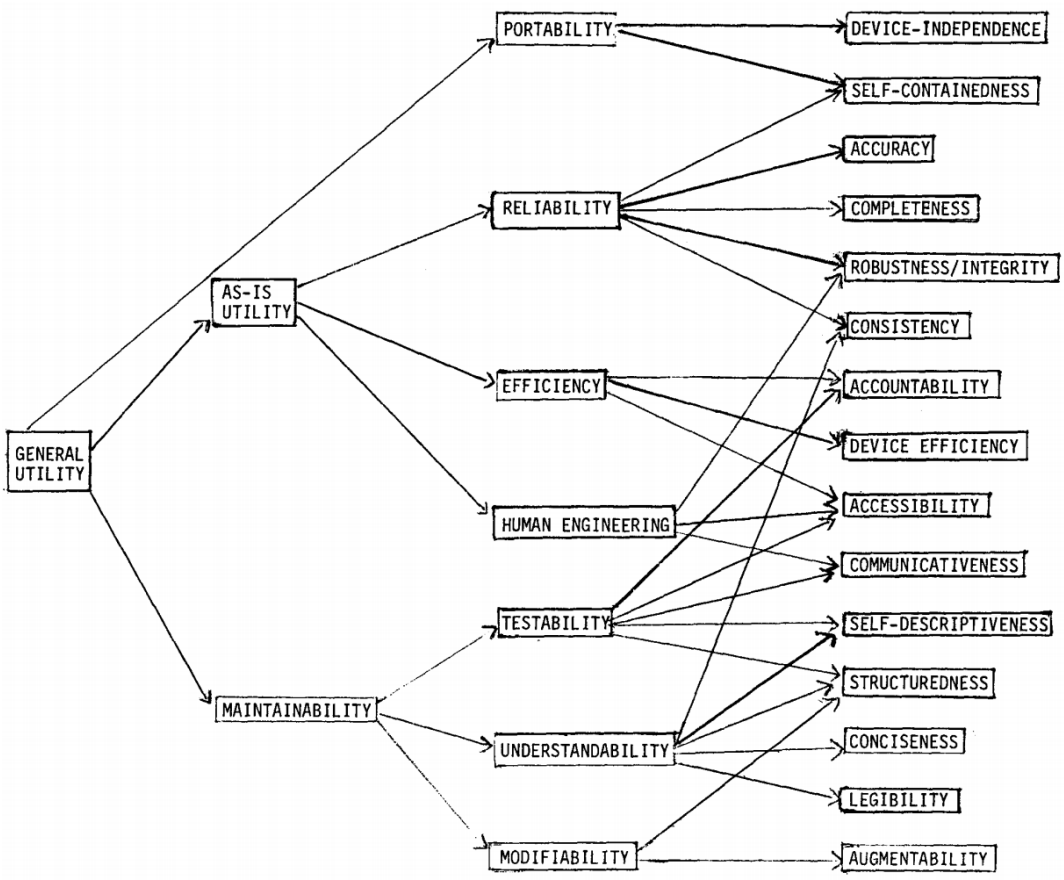


Figure 10: An overview of the quality model proposed by Boehm et al. [BBL76].

2. Reusability: how easy or difficult can a component be used in other systems?

3. Analysability: how easy or difficult is it to diagnose which components of a software should be modified and what the impact is on the software product?

4. Modifiability: how easy or difficult can the software product be changed without causing defects or a lower software quality?

5. Testability: how easy or difficult can the software product be tested on several test criteria after a modification?

4.2 Methods for creating a quality model

4.2.1 Goal-question-metric approach

According to a study by Rombach et al. [RB90], it is important to have measurable goals and questions in order to create an effective model. In 1984 Basili et al. proposed the so-called goal-question-metric approach (GGM) [CR94] which is used for defining measurement goals. They indicate that measurements should be defined using a top-down approach instead of a bottom-up approach. There are to many metrics that can be used to characterize a software such as lines of

Figure 11: An overview of the quality model proposed by International Standards Organization (ISO) [iso].

code, complexity, number of defects and many more. By using a bottom-up approach, it is difficult to say which of these metrics should best be used. A top-down approach will mitigate this problem because goals and questions are first defined before the metrics are chosen.

The goal-question-metric approach consists of three levels which are:

1. **Goals** The goal-question-metric approach makes the assumption that it is important to first specify the goals of the quality model such that there is a purpose for the measurements. A goal consists of four elements which are: Purpose of measurements, issue to be measured, object to be measured and viewpoint from which the measure is taken.

2. **Questions** The second hierarchical level are the questions which should indicate how the defined goals can be assessed and thereby indicate whether or not a specific goal is met.

3. **Metrics** The third level are the metrics which should indicate how the defined questions can be measured in a quantitative way. These metrics could be either subjective or objective.

An overview of the hierarchical structure of the goal-question-metric approach can be found in Figure 12. This approach has already been used by many studies for (1) creating a quality model for extending software quality assessment techniques to java systems [PMDL99]; (2) creating a model for measuring the understandability and maintainability of BPMs [GWMW08]; (3) creating a model for measuring the quality in the application development sector [SRE18]. The goal-question-metric approach can be used for defining GGM models for software quality improvement tasks which could therefore be used as the approach for developing a quality model in this master thesis.

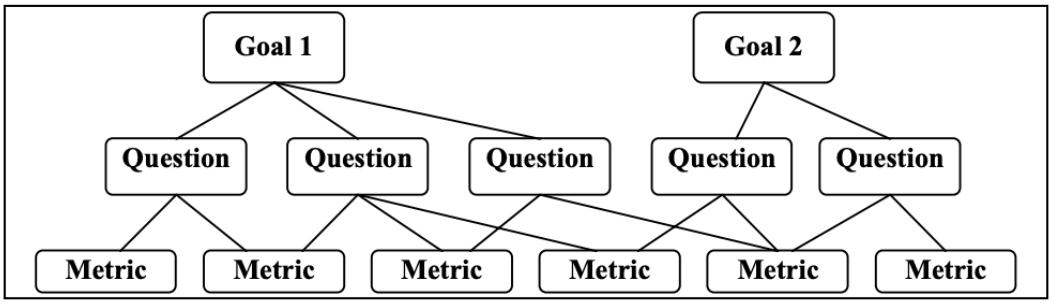


Figure 12: An overview of the goal-question-metric approach proposed by Basili et al. [CR94].

4.2.2 Factor-criteria-metric approach

The well-known quality models developed by Boehm [BBL76] and McCall [MRW77], explained in the previous section, are not based on the goal-question-metric approach [CR94] but on the factor-criteria-metric approach [MR04]. Similar to the goal-question-metric approach, it is constructed using a tree structure containing three levels which are:

1. **Factor** The factor-criteria-metric approach indicates that one should start with defining the high-level quality factors such as maintainability or usability which one want to measure.

2. **Criteria** The second hierarchical level are the criteria which should divide the quality factors into smaller pieces that are easier to understand and measure. By measuring the criteria, we can measure the high-level quality criteria as the hierarchical tree structure suggest.

3. **Metrics** The third level are the metrics which should indicate how the defined criteria can be measured in a quantitative way.

Although this approach has been used by many studies [BBL76][MRW77], there are two major drawbacks according to Marinescu and Ratiu [MR04]. First of all, it is difficult to trace-back how the quality criteria are mapped onto the metrics. The mapping is based on a set of rules and practices of good-design but this is not recorded anywhere in the model. Secondly, a quality model based on the factor-criteria-metric approach can not be used to find the issues causing a lower software quality.

5 Research methodology

This chapter will briefly explain the methodology that has been used to answer the research questions mentioned in Section 1. This chapter will outline the steps taken and methods used to collect and analyze the data in order to create an answer to the research questions. Furthermore, the reasons behind the chosen steps and methods are explained.

5.1 Research overview

The goal of this master thesis is to develop a quality model that contains product metrics that can be used to measure the maintainability of a Splunk app. There are two main deliverables of this master thesis which are a developed quality model and a tool to measure and visualize the maintainability score of each component of a Splunk app. In order to realize these deliverables, we went through the following steps (1) We first created an initial quality model which will be validated by conducting two focus groups and a survey; (2) After implementing the results of the focus groups and survey, we will create a benchmark rating system for the product metrics in our quality model; (3) The final version of the quality model will be translated into a design of the tool which will be validated by conducting an empirical investigation at SMT.

In this master thesis, we will be using the design science research method developed by Offermann, Levina, Schönherr, and Bub [OLSB09]. The proposed research process consists of three main phases: "problem identification", "solution design" and "evaluation". In the problem identification phase, we will conduct literature research to get an understanding of the problem. In the solution design phase, we will design an artifact which is a quality model supported by literature research and experts. In the evaluation phase, we will evaluate the effectiveness of the proposed solution by conducting a case study and an expert survey [OLSB09].

5.2 Research approach

As mentioned in the previous paragraph, this master thesis can be divided into five different steps that eventually lead to the desired deliverables.

5.2.1 Phase 1: Create initial quality model

In order to develop a quality model for Splunk apps, we have to conduct a theoretical investigation into the Splunk language in order to get a detailed overview of the various concepts and how they work within the Splunk language. This is also important for defining the metrics because each component of a Splunk app should be investigated in order to define the most relevant metrics for that particular component. After that, we have developed an initial quality model by conducting literature research in combination with the knowledge and experience of the thesis author as a Splunk developer. Because the initial quality model is created by using the literature in combination with the knowledge and experience of the thesis author as a Splunk developer, we have to validate this model which will briefly be explained in the next section. In order to develop the quality model, we have used the goal-question-metric approach which is briefly explained in Section 4.

First of all, I have defined the main goal of the quality model which has been sub-divided into the sub-goals that specifies the reason behind the measurement. After defining the goals we have to refine each goal into several questions that break the issue, defined in the goal, down into the major components. These questions will indicate how the goals can be evaluated whether those goals are met by taking into account the selected viewpoint defined in the goal. After defining the questions we have to refine each question into several metrics that indicate how the questions can be answered [CR94]. Here, it is possible that a metric can be used for answering multiple questions. For the amount of metrics, we have used the MECE principle which is the abbreviation for "mutually exclusive and collectively exhaustive" developed by Barbara Minto [Min09]. This means that we don't have to define as many metrics as possible, but we have to define metrics such that it has a high coverage but as little overlap as possible. In this research project, it means that the metrics should answer all the defined questions but should not measure the same aspect of the code.

For defining the metrics, we have separated our quality model into the two languages which are the Python programming language and Splunk. After that, we conducted literature research for defining code level metrics that can be used to measure the maintainability of Python code. By using our obtained knowledge about the Splunk language, we have tried to translate the Python related metrics to metrics that can be used within the Splunk language. Furthermore, we have added some metrics to the Python and Splunk list by using the knowledge and experience of the thesis author as a Splunk developer.

5.2.2 Phase 2: Validate the quality model

The initial quality model has been developed by conducting literature research in combination with the knowledge and experience of the thesis author as a Splunk developer and should therefore be validated. In order to validate the initial quality model, we will conduct two focus groups in combination with a survey as this is recommended in the literature [KBL08]. Both research methods have a different goal and validate a different part of the initial quality model.

Focus group The focus groups have the goal to validate the combination between the goals and the questions of the quality model. This could give me answers to the questions: Are these the right questions for evaluating the defined goals? Are the goals and questions easy to understand? This could lead to the deletion or addition of questions indicated by the participants of the focus groups. Furthermore, the input of the participants could also lead to the addition or deletion of product metrics defined in the initial quality model. The results of the focus groups will be implemented in the quality model resulting in a new version of the quality model. For validating the combination between the goals and the questions, we have conducted two focus groups with 4 to 6 Splunk developers. This is also a limitation of my qualitative research as research has shown that, preferably, you should do 4 to 6 focus groups consisting of 3 to 12 participants [KBL08]. The focus group sessions took place in an online meeting which have been recorded such that I could analyse the input of the participants after the session. Furthermore, literature also mentioned that the moderator of the focus group does not have the time to write everything down during the session because he or she has to manage the session [KBL08].

The focus group session has been divided into three parts which are: Pre-session, during-session and post-session. In the pre-session part, we have explained the idea behind the focus group that will be discussed in more detail during the session. This will ensure that the session time is used most effectively for discussions because the participants already get a general idea about the topic and have thought about the topic [KBL08]. During the focus group session, we followed the following structure:

- First of all, the session was initiated by an introduction where the goals and ground rules of the session were also explained to the participants. During this introduction, I have introduced the topic in order to get on the same page and refresh the memories of the participants. During this step, I have explained the topic and gave an explanation of some fundamental concepts.

- After that, I have explained the defined goals of my quality model which could then be criticised by the participants. This could lead to interesting information about how the participants think of the defined goals such as:

  1. Are the goals easy to understand?
  2. Are the goals relevant for measuring maintainability?
  3. Are there goals missing?

- Then, I have asked the participants which questions he or she would define in order to evaluate whether those goals are met without mentioning the questions I have defined in the initial quality model. This could lead to interesting information such as:

  1. Which questions the participants think are important to evaluate whether the defined goals are met without narrowing the view of the participants.
  2. Confirmation whether the participants indeed understand the goals.

- After that, I have showed and explained the questions defined per goal in the initial quality model which we have then compared with the questions defined by the participants. This could lead to interesting information about how the participants think of the defined questions such as:

  1. Are the questions easy to understand?
  2. Are the questions relevant for making the goals measurable?

- Lastly, I have asked the participants some general questions about maintainability regarding Splunk apps. For example, what is the biggest problem of understanding the code of other Splunk apps?

The schedule of the focus group can also be found in Appendix A. Lastly, in the post-session part I have analysed the recordings and wrote down the important information mentioned by the participants. The results of the focus group sessions have then been implemented in the quality model creating a new version of my quality model. After implementing the results of the focus group sessions in the quality model, I was able to start with the second validation.

Survey The survey has the goal to validate the combination between the questions and the metrics of the second version of the quality model. This could give me answers to the questions: Are these the right metrics for evaluating the defined questions? Are the questions and metrics easy to understand? This could lead to the deletion or addition of metrics indicated by the participants of the survey. The results of the survey will be implemented in the quality model resulting in a new version of the quality model. For validating the combination between the questions and the metrics, we have conducted a survey with 17 participants. The participants of the survey are either Splunk developers or they have experience in modifying/creating Splunk apps.

The structure of the survey has been divided into three parts which are: introduction, specific and general. In the introduction, we have explained the goal of the survey and gave a small introduction to the topic. In the specific part, we asked questions in order to validate the metrics defined for each question in our quality model. In the general part, we asked a few questions regarding the maintainability of Splunk apps in general. There were roughly two types of questions, open question and multiple-choice questions. The goal of the open questions was to receive additional metrics or feedback per question of our quality model. The multiple-choice questions were asked by using the 5-point Likert-type scale which requires the participant to indicate whether he or she strongly disagrees, disagrees, is neutral, agrees or strongly agrees with a certain statement. The statements were all positively-worded as Barnette found that the internal consistency is then higher compared to a survey containing negatively-worded or a mixture of negatively and positively-worded statements [Bar00]. The statements of the survey are discussed and can be found in Section 7.

5.2.3 Phase 3: Finalize the quality model

After implementing the results derived from the focus group sessions and survey, a validated version of the quality model has been created. This quality model contains code-level metrics that have been indicated as relevant by the participants of the survey and focus groups. We have finalized the quality model by using a benchmark data set to verify whether the metrics give useful information and therefore is a good or bad candidate for measuring the maintainability of a Splunk app. The benchmark data set contains the code of 50 public Splunk apps which is briefly explained in Section 10.

In order to verify whether the metrics give useful information and is therefore a good or bad candidate for measuring the maintainability of a Splunk app, we have used the following approaches. We assessed the usability of the metrics by looking at whether some metrics are only present in a small number of Splunk apps and how the metric values are distributed using the metric values derived from the benchmark data set. This assessment could also lead to the deletion of some metrics which results in a final set of metrics for our final quality model.

After deriving the final set of metrics, we will finalize our quality model by developing a benchmark rating system for each metric in our quality model. These rating systems will be used to assess new Splunk apps by comparing the metric values of the new Splunk app with the values derived from the benchmark. In Section 2.4, we have briefly explained how the benchmark rating systems are derived.

5.2.4 Phase 4: Translate the quality model into a tool

After creating the final version of the quality model, we have translated the obtained quality model into a design for the Splunk app that can be used to continuously measure and visualize the maintainability score of a Splunk app that is being developed. We have only implemented the Python scripts for measuring the metrics defined for the volume question in our final quality model which are explained in Section 9. The metric and rating thresholds are also only defined for these metrics in order to indicate the feasibility of this model which are explained in Section 10.2.

5.2.5 Phase 5: Validate the model at SMT

The derived model and the design of the tool, which is a Splunk app, have then been evaluated by conducting an empirical investigation at SMT. This has been done by conducting three interviews with Splunk developers of SMT which results are discussed in Section 11.

6 Create initial quality model

In this chapter, we will explain how we derived the initial quality model containing quality metrics that will be measured in order to calculate the maintainability score of Splunk apps. This chapter will outline the three different aspects of the goal-question-metric approach, which is briefly discussed in Section 4.2.1. We have developed the initial quality model by conducting literature research in combination with the knowledge and experience of the thesis author as a Splunk developer. Because the initial quality model is created by using the literature in combination with the knowledge and experience of the thesis author as a Splunk developer, we have validated this model by conducting two focus groups, a survey (7) and we have analyzed the performance of the metrics defined for the Splunk language as they have not yet been analyzed in the literature 8.

6.1 Initial quality model: Goals

The first step of the goal-question-metric approach is to define the goals of the quality model which indicate what the users will achieve with this quality model. First of all, we have defined the main goal which is then sub-divided into the goals which specifies the reason behind measuring the maintainability of Splunk apps. Here, the goals consist of 4 elements which are purpose, object, issue and viewpoint. The main goal is defined as follows:

> *The overall goal of this research project is to improve the control over the maintainability costs from the CEO's viewpoint.*

This main goal can be divided into four sub-goals using the definition of maintainability according to Boehm's Quality Model (4.1.2) and ISO (4.1.3). In this quality model, maintainability should give answers to the questions: how easy is it to understand, modify and retest the software product? In Table 1, the defined goals are divided into these 3 elements.

Goal	**Purpose**	**Issue**	**Object**	**Viewpoint**
Improve the speed to **understand** the software from the development teams' viewpoint	Improve	The speed to understand	The software	Development team
Improve the easiness to **test** the software from the development teams' viewpoint	Improve	The easiness to test	The software	Development team
Increase the effectiveness and efficiency to **modify** the software from the development teams' viewpoint	Increase	The effectiveness and efficiency to modify	The software	Development team

Table 1: The defined goals using the structure of the goal-question-metric approach.

In order to improve the control over the maintainability costs, the CEO should delegate it to the development team because they have a direct influence on the maintainability costs. Therefore, the viewpoint of the sub-goals have changed from the CEO's viewpoint to the development teams' viewpoint compared to the main goal, as shown in Table 1. The degree of understandability of the source code, how much time and effort would it take to understand the source code, has a direct impact on several software development tasks. As mentioned by [MBWW20], professional

developers spend approximately more than 50% of their time on tasks related to understanding the source code. For example, the understandability of the source code has an impact on the time and effort needed (i) to fix a bug (the lower the understandability, the more time and effort needed to fix a bug); (ii) to perform code reviews after, for example, a modification (the lower the understandability, the more time and effort needed to review the code) [SBV+17].

The degree of testability of the source code, how much time and effort would it take to test whether the source code meets the requirements and to identify defects/bugs, has a direct impact on the activities of a development team. For example, the testability of the source code has an impact on the time and effort needed (i) to identify a bug or defect (the lower the testability, the more time and effort needed to identify a bug or defect); (ii) to deliver a high quality software product with little bugs [BOB+20]. The degree of modifiability of the source code, how much time and effort would it take to change or expand the source code, has a direct impact on the activities of a development team. As mentioned by [YC88], the modifiability of software products takes up most of the maintenance time and effort.

6.2 Initial quality model: Questions

After defining the goals, we have to refine each goal into several questions that break the issue, defined in the goal, down into the major components. These questions will indicate how the goals can be measured by taking into account the selected viewpoint defined in the goal. In Table 2, 3 and 4, the defined questions for the goal about understandability, testability and modifiability respectively have been specified.

Question	**Motivation**
To what extent do you comply with the **coding guidelines** that guide you how to structure the code of your Splunk app?	By using clear requirements or coding guidelines, developers not familiar with the project can better understand the code.
What is the **volume** of the code? [HKV07]	According to the SIG quality model, the total size of a system should feature heavily in any measure of maintainability. A larger system requires, in general, a larger effort to maintain. Furthermore, the larger a function or file is, the more actions/functionalities it contains and thus the more difficult it is to understand that function or file.
What is the **readability** of the code? [BW09] [SBV+17]	The readability of a program is related to its maintainability, and is thus a key factor in overall software quality. Furthermore, a low readability will also negatively affect the understandability of the code.
Does a clear **documentation** exist for the software? [SBV+17]	A good and clear documentation will help maintainers, who are not familiar with the software, to understand the software and code by explaining the functionalities and different parts in the code.

Table 2: The defined questions for the first goal (understandability) including the motivation of defining this question is specified in this table.

Question	Motivation
Is the **logging** instrumentation inplace?	One of Splunk's most important applications is the processing of log data and use it to create better insights into, for example, the performance of code. Furthermore, it will lower the MTTR because the logs give a good indication of the location of the problem and the problem itself.
What is the **volume** of the code? [HKV07]	According to the SIG quality model, the total size of a system also affects the testability of a system. Furthermore, the larger a function or file is, the more actions/functionalities it contains and thus the more difficult it is to test all possibilities of that function or file.
What is the **complexity** of the source code? [HKV07]	According to the SIG quality model, the complexity of the source code units influences the system's changeability and its testability.

Table 3: The defined questions for the first goal (testability) including the motivation of defining this question is specified in this table.

Question	Motivation
How well is the code **structured**? [BBL76]	Modifiability consists of the following sub-characteristics: 1. Structuredness 2. Augmentability A more structured Splunk app will increase the ease to modify that Splunk app.
What is the amount of **duplication** in the code? [HKV07]	According to the SIG quality model, the degree of source code duplication (also called code cloning) influences analysability and changeability. Furthermore, the more duplication in the code, the more time it takes to modify the code because the changes have to be made to all duplicate code fragments.
What is the **complexity** of the source code? [HKV07]	According to the SIG quality model, the complexity of the source code units influences the system's changeability and its testability.
What is the **volume** of the source code?	The larger a file or unit is, the more difficult it is to modify that unit or file because the changes have effect on multiple fragments of that function or file.
What is the amount of **hardcodes** in the source code?	The more hardcodes, the more difficult it is to modify the source code because it is difficult to know where you have to make some changes.

Table 4: The defined questions for the first goal (modifiability) including the motivation of defining this question is specified in this table.

The questions are defined by using the literature in combination with the knowledge and experience of the thesis author about how we can make those goals measurable. In the table, we have indicated questions that are defined by using the literature by mentioning the source from which we derived

the question. By answering these questions, we should be able to give an indication whether the goals are met.

6.3 Initial quality model: Metrics

After defining the questions, we have to refine each question into several metrics that indicate how the questions can be answered/measured by taking into account the selected viewpoint defined in the goal. A metric can be used for answering multiple questions under the same goal. Here, we use the MECE principle which is the abbreviation for "mutually exclusive and collectively exhaustive". This means that we don't have to define as many metrics as possible, but we have to define metrics such that it has a high coverage but as little overlap as possible. For each of the quality metrics, thresholds should be defined that indicate how bad it is by using the methodology explained in Section 2.4. In Table 5, 6 and 7, the defined metrics for each question for the goal about understandability, testability and modifiability respectively have been specified separated in Python and Splunk metrics. In some cases, the metric for Python and Splunk are the same.

The metrics are defined by using the literature [ACBV20][SR18][SLVPO16][CH17][BW09][VDBvdL07], in combination with the knowledge and experience of the thesis author about how we can make those questions measurable. We have created a full list of possible code-level metrics which has not yet been filtered based on e.g. literature, measurability or whether the output is useful. We have created a full, unfiltered list of code-level metrics because we first want to receive the opinions of Splunk developers regarding those metrics by conducting two focus groups and a survey which will be explained in Section 7. In Section 8, we will create a final, filtered list of code-level metrics which will be further discussed in that chapter.

Question	Metric Python	Metric Splunk
To what extent do you comply with the **coding guidelines** that guide you how to structure the code of your Splunk app?	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
	# of descriptive variable names (use of nouns)	# of descriptive variable names
	# of descriptive function names (use of verbs)	# of descriptive panel names
	# of comment lines in the code (CLOC)	# of comment lines in the code
		# of compliance errors with Splunk app structure
What is the **volume** of the code?	# of tokens in a function	# of tokens in a SPL query
	# of tokens in a file	# of tokens in a file
	# of tokens in a Splunk app	# of tokens in a Splunk app
	# of files in a Splunk app	# of files in a Splunk app
	# of statements in a function	# of commands in a SPL query
	Tiobe fan-out score of a file	# of drilldowns
	# of methods in a file	# of views
		# of stanzas in a file
		# of Splunk knowledge objects
		# of fields used in a SPL search
		# of data sources used in a SPL search
What is the **readability** of the code?	# of tokens in a file	# of tokens in a file
	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
	# of descriptive variable names (use of nouns)	# of descriptive variable names
	# of descriptive function names (use of verbs)	# of descriptive panel names
	# of characters on a single line	# of characters on a single line
	# of indentation on a single line	# of indentation on a single line
	# of nested statements in a function	# of nested searches in a SPL query
	# of identifiers in a file	# of commands in SPL query
	# of '{' and '(' characters on a single line	# of {' and '(' characters on a single line in SPL query
	# of '.' characters in a function	
	Cohesion of a method	
Does a clear **documentation** exist for the software?	Gunning Fog Index of the documentation	Gunning Fog Index of the documentation
	The existence of a documentation (true, false)	The existence of a documentation (true, false)
	Comments and Identifiers Consistency in a file	Comments and Identifiers Consistency in a file
	Documentation and Identifiers Consistency in a file	Documentation and Identifiers Consistency in a file

Table 5: The defined metrics per question for the goal about understandability are specified in this table.

Question	Metric Python	Metric Splunk
Is the **logging** instrumentation inplace?	# of logging statements per function	The existence of a monitoring dashboard
What is the **volume** of the code?	Same metrics as in first question	Same metrics as in first question
What is the **complexity** of the source code?	Same metrics as in third question	Same metrics as in third question

Table 6: The defined metrics per question for the goal about testability are specified in this table.

Question	Metric Python	Metric Splunk
How well is the code **structured**?	# of tokens in a function	# of tokens in a SPL query
	# of tokens in a file	# of tokens in a file
	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
		% of compliance with Splunk app structure (Using AppInspect)
What is the amount of **duplication** in the code?	Tiobe duplication score	# of duplication between SPL queries on a dashboard or config file
What is the **complexity** of the source code?	Cyclomatic complexity of a function	# of input data fields used in a SPL query
	Tiobe fan-out score of a file	# of data tables used in a SPL query
	Cohesion of a method	# of output data fields used in a SPL query
	Coupling in a file	# of sourcetypes used in a SPL query
		# of data points used in a SPL query
What is the **volume** of the source code?	Same metrics as in first question	Same metrics as in first question
What is the amount of **hardcodes** in the source code?	# of hardcoded variables	# of hardcoded variables

Table 7: The defined metrics per question for the goal about modifiability are specified in this table.

7 A focus group and survey on maintainability of Splunk apps

In this chapter, we will explain how we have validated the initial quality model described in Section 6. Because the initial quality model is created by using the literature in combination with the knowledge and experience of the thesis author as a Splunk developer, we have validated this model by conducting two focus groups and a survey which are explained in this section.

7.1 Focus group

According to [KBL08], empirical research methods have received an increased interest in the software engineering field. An example of such an empirical research method is the so-called focus group which is a qualitative research method. In 1996, Morgan defines focus groups as a "research technique that collects data through group interaction on a topic determined by the researcher" [Mor96]. This means that (i) a focus group can be used for data collection (ii) group discussions are the main data source of a focus group and (iii) the person who is conducting the focus group is responsible for creating the desired group discussions [Mor96]. The main benefits of conducting a focus group is that they produce useful information without spending a lot of resources such as time [WHW91].

The first validation of the initial quality model has been done by conducting two focus groups. The first focus group session took place on Wednesday 26th of May from 13:00 till 16:00 o'clock with 4 participants. The majority of the participants had a background in Splunk development or Splunk implementation and have been part of a development team in which they developed a Splunk app in the past 6 months. The second focus group session took place on Thursday 27th of May from 10:00 till 12:00 o'clock with 4 participants. The majority of the participants had also a background in Splunk development or Splunk implementation and have been part of a development team in which they developed a Splunk app in the past 6 months. According to [KBL08], a focus group should consist of around 3 to 12 participants which means that we have met this requirement by having 4 participants in each focus group.

A focus group should be guided and facilitated by the person who is conducting the focus group, and should follow a predefined questioning structure in order to keep the discussions on the right track. A focus group consists of the following four steps [KBL08]:

1. Planning the Research: it is important to define the contribution of the focus group in your research process. In this master thesis, the goal of the focus groups is to validate the combination between the goals and the questions defined in the initial quality model. This could give me answers to the questions: Are these the right questions for making the goals measurable? When I have the following goals, are these the appropriate questions to measure those goals? This could lead to the deletion or addition of questions indicated by the participants. The results of the focus group will be implemented in the quality model resulting in a new version of the quality model.

2. Designing focus groups: after defining the role of the focus group in the research process, it is important to select the right participants. According to [KBL08], selecting the right

participants is crucial for a successful focus group. For our focus groups, we have selected motivated and enthusiastic participants having a background in Splunk development or Splunk implementation and have been part of a development team in which they developed a Splunk app in the past 6 months. Furthermore, segmentation is an important aspect of designing a focus group which refers to different compositions of the group. For our focus groups, we have created two different groups:

- The first group consisted of participants who have spent a lot of their time developing or analysing a Splunk app.
- The second group consisted of participants who spend most of their time working with the Splunk apps or are a product owner in a project for creating a Splunk app instead of developing the Splunk app themselves.

3. Conducting the focus group sessions: a focus group usually takes up to 2 or 3 hours which follows a predefined questioning structure. It is important that the person who is conducting the focus group, facilitates and manages the discussion in the right direction. For our focus group, we have created a schedule and structure which can be found in Appendix A. Furthermore, the focus group session took place in an online meeting which has been recorded such that we could analyse the data after the session.

4. Analyzing the data and reporting the results: the recordings of our focus groups have been transcribed such that we were able to summarize the answers of the participants on our questions. In the next sub-section, we will further discuss the results of our focus groups.

7.1.1 Results

As mentioned in Section 7.1, we have conducted two focus groups with two different compositions in terms of the characteristics of the participants. The research by [KBL08] mentioned that groups with different compositions could lead to different kind of answers. This was also an interesting observation in our focus groups where the participants of one of the groups were thinking on code level, visualizing that they are developing a Splunk app and the other group was more thinking in terms of using a Splunk app instead of developing a Splunk app.

The goal of the focus groups was to validate the combination between the goals and the questions defined in the initial quality model. What do the participants think of the defined goals? What do the participants think of the defined questions and its combination with the goals?
Goals of the quality model After explaining the structure and goals of the focus group and giving an introduction to the topic, we started with a discussion about the goals of the quality model. We first asked the participants how they would define the maintainability of Splunk apps and why they think it is important to measure and improve the maintainability. The terms that the participants linked to the maintainability of Splunk apps were: modifiability, understandability, testability, complexity and structeredness. Furthermore, they indicated that how easily someone can solve bugs and whether a version control system was present are also part of maintainability. One of the participants of the focus group mentioned the following which is related to modifiability:

When he or she developed a Splunk app and someone else, not part of the development team for that Splunk app, can add a new feature to the Splunk app within a reasonable timeframe and without a lot of effort, the Splunk app has a high maintainability.

Another participant also mentioned the following which is related to understandability:

It is important that someone else should know where to make certain changes in the code, for example, to add a new feature or to fix a bug.

The following was also mentioned by the partcipants which is related to testability:

It is very important that someone can easily and quickly detect the root cause of a bug such that the bug can be solved within a short period. This could be achieved by having good log messages that indicate whether and why a certain functionality failed or succeeded, which should also be visible in Splunk, for example in the "_internal" index.

In general, the participants of both groups agreed with the defined goals which are related to understandability, testability and modifiability. There was also one participant who mentioned that it could be useful to measure whether the output of the app is correct:

It could also be helpful to measure the output of the app whether it is correct. Because the understandability, modifiability and testability could be very good, but the Splunk app isn't very useful if the output is not correct.

However, this would be part of another quality characteristic according to the ISO/IEC 25010 [iso] quality model which is functional suitability and is therefore not taken into account in this master thesis.

Questions of the quality model After the discussion about the goals of the quality model, we started another discussion about the questions of the quality model. We asked the participants which questions they would define in order to make the goals measurable and thereafter about whether they agreed with the questions defined in the initial quality model. The main topics of the questions per goal defined by the participants were as follows:

1. Understandability: Documentation, comments, coding standards, volume, visualisation of the data flows
2. Testability: Time to test, are the configurations on one place, logging, unit or functional tests, sample data, external dependencies
3. Modifiability: Volume, coding standards, visualisation of the data flows, comments, logging

Most of the topics indicated by the participants are indeed included in our initial quality model. After asking the participants to formulate potential questions, we have explained the questions we defined in the initial quality model. The participants mentioned that the questions defined for the first goal, about understandability, are understandable and complete.

In general, the participants mentioned that the questions defined for the second goal, about testability, are also understandable and complete. However, the participants mentioned the following regarding the questions for testability:

It could be useful to check whether a monitoring dashboard is available and whether the system paths are hardcoded in each Python file or centralized in a single config file such that it only has to be changed in one place.

Furthermore, one participant indicated that the amount of external dependencies has also an impact on the testability of a Splunk app.

What are the external dependencies? This causes more external systems to be configured and available in order to test the Splunk app. For example, when the code makes a connection between a mail server, a mail server should be available in order to test this functionality.

In general, the participants mentioned that the questions defined for the third goal, about modifiability, are also understandable and complete. However, the participants mentioned the following regarding the questions for modifiability:

It could be useful to check whether the code is generic for different operating systems.

Furthermore, the participants mentioned that the amount of external dependencies has also an impact on the modifiability of the source code. And one participant also indicated that the existence of a CICD pipeline would also be an useful check.

Does a CICD pipeline exist? Because when a CICD pipeline is available, the maintainability will be higher because it is easier to make changes to the source code or to add new features. Furthermore, this gives you the ability to easily check which change broke the Splunk app.

Lastly, the participants indicated that it might be useful to change the development team's viewpoint to a broader viewpoint such that the consultants, who install the app by the customers, are also taken into account. However, the main application of our quality model is to facilitate the developers with factual data about the maintainability of a Splunk app they are developing. This means that our tool/quality model will be used during the development phase of a Splunk app to make well-informed decisions about, for example, whether or not they should refactor their source code in order to increase the maintainability.

General questions Lastly, we started a discussion about more general questions such as what is the biggest problem you have encountered during your work which was associated with the understandability of a Splunk app?; Which aspects of a Splunk app makes a Splunk app complex? One participant indicated that the amount of custom Python files and knowledge objects makes a Splunk app more complex. Furthermore, the participants mentioned that no comments in a SPL query and the length of a SPL query makes it more difficult to understand. It was also mentioned that the complexity is very important for the understandability of a Splunk app.

The complexity of a Splunk app is the biggest factor for determining whether a Splunk app is easy to understand. When a Splunk app contains a lot of different Python files which are all coherent, it is difficult to understand how the Splunk app actually works.

Lastly, the participants mentioned that the number of data points in a SPL query and the number of tokens on a dashboard has a direct impact on the complexity of a Splunk app.

7.2 Survey

The first validation of our initial quality model has been done by using the focus group method which goal was to validate the defined goals and questions in the model. However, according to Morgan [Mor96], the use of the focus group method is often combined with other research methods such as a survey. It is mentioned that there are four ways to combine focus groups with a survey. One of the four combinations is using a survey as the primary research method in which the focus groups are used to provide information that can be used to optimize and supplement the survey questions [Mor96].

By using this information, we did a second validation by conducting a survey. The goal of this survey was to validate the defined questions and metrics in the initial quality model. We have chosen to validate only the metrics defined for the Splunk language as the metrics for the Python language are already validated by other literature. This could give us answers to the questions: Are these the right metrics to measure for answering the defined questions? This could lead to the deletion or addition of metrics indicated by the participants. The survey has been developed according to the following steps:

1. First of all, we have created a first version of our survey which has been optimized and supplemented by using the obtained information from the focus groups. In order to validate the understandability of the survey and whether the answers are as expected, we have conducted a pilot version to obtain feedback which has been used to improve the survey. During this pilot version, we were able to observe the participant via an online meeting. It was mentioned by the participant that the formulation of the questions was hard to understand and that the survey contains to much text/explanation.

2. Secondly, we have implemented the feedback of the participant from our first pilot session. In order to validate the new formulation of questions, we left one of the questions as it was formulated in our first pilot session. In the second pilot session, the participant immediately indicated that the new formulation of questions was much better to understand as the participant didn't understand the old formulated question. Furthermore, the participant indicated that the code snippets were difficult to understand as it is important to know the context of, for example, a SPL query.

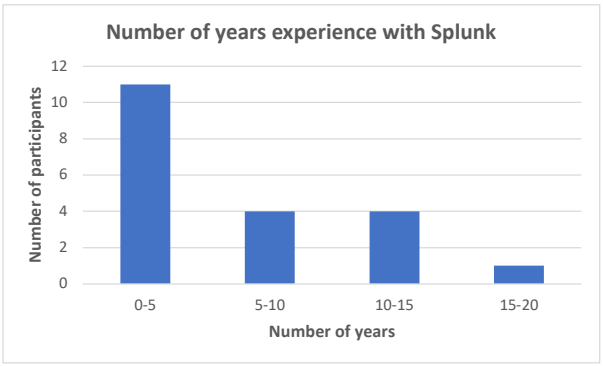
3. Lastly, we have implemented the feedback of the participant from our second pilot session. This included changing the formulation of each question using the new type of formulation and removing the code snippets except for one code snippet which has been used to define a coding standard regarding the layout of a SPL query. The participant of the third pilot session mentioned that the survey was easy to understand.

After the aforementioned steps, we have analysed the output of the survey in order to check whether it was correct and usable. The output of the survey was as expected after which we have distributed the survey. The survey has been distributed on multiple channels such as LinkedIn, Reddit, Splunk slack group and several companies doing Splunk app development. The target audience for the survey were Splunk developers or people who has experience with modifying or creating a Splunk app. This is also verified in the last few questions in which we asked the participants about their

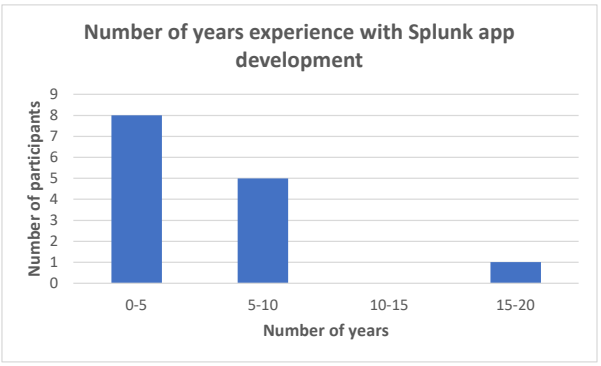
experiences and contributions to Splunk app development. The survey was available from the 21st of June until the 5th of July.

7.2.1 Results

As mentioned in 7.2, we have conducted a survey among Splunk developers or people who have experience with modifying or creating a Splunk app. At the end of the survey, we asked the participants about their experience with Splunk and Splunk app development. In total, 17 persons have participated in our survey having an average experience with Splunk of 6.38 years and 4.35 years with Splunk app development. The number of years experience with Splunk varied from 1 to 16 years which has been plotted in Figure 13a. The number of years experience with Splunk app development also varied from 1 to 16 years as shown in Figure 13b.



(a) Number of years experience with Splunk



(b) Number of years experience with Splunk app development

Figure 13: The experiences of the participants regarding Splunk and Splunk app development. The participants had an average of 6.38 years experience with Splunk. And the participants had an average of 4.35 years experience with Splunk app development. In total, 17 persons have participated in our survey.

There are a number of factors that could have caused this relative small amount of participants (17). First of all, the segment for our participants is narrowed to only Splunk app developers or people who have experience with modifying or creating a Splunk app. Secondly, after reaching out to possible participants, they must be willing to spend between 20 to 30 minutes for filling in the entire survey. However, besides the low number of participants of the survey, they have provided us with valuable and reliable data as the average time spent completing the survey is around 1 hour and 22 minutes. Furthermore, they also left valuable information in the open questions which have let to the addition of new metrics or the deletion of old metrics.

Now, we will discuss the results of the survey questions separated in the same blocks as in the survey. Each block in the survey represented a question defined in the quality model. In each block, we asked the participants to give their opinion about a number of statements that could vary from 2 to 13 statements depending on the block. Each statement in a block represented a Splunk metric defined in our quality model and could be answered according to the 5-point Likert scale

consisting of the options: totally disagree, disagree, neutral, agree, totally agree. An example of a statement question is shown in Figure 14. The answers to the statements will be used to select the Splunk quality metrics for our final version of the quality model discussed in Section 8.

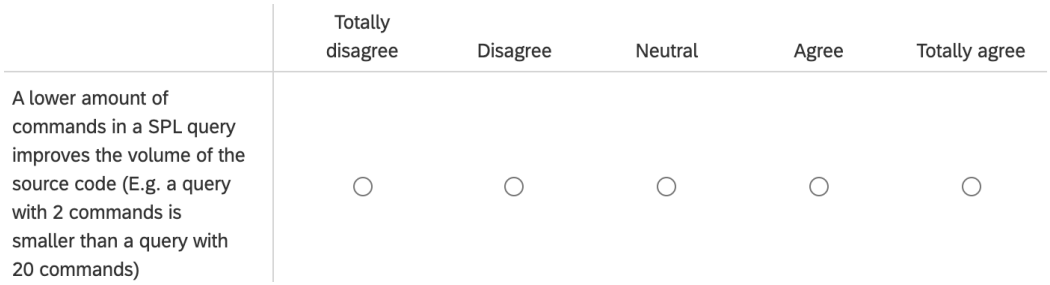


Figure 14: An example of a statement defined in the multiple-choice questions of the survey.

We have calculated the average score of each statement by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, we have also calculated the total average score per question by using the answers on the statements that belongs to that question. However, our survey consists of stand-alone Likert questions which means that we have Likert-type items instead of Likert scales. Likert-type items are part of the ordinal measurement scale which means that it is not recommended to use averages. Therefore, we have also created a bar-chart containing the frequencies of each Likert scale answer per statement. This bar-chart will be used in combination with the averages to add more substance to these average values [BB12]. Each bar-chart has been ranked from most agree to least agree based on the frequencies of totally disagree, disagree, neutral, agree and totally agree.

7.2.1.1 To what extent do you comply with the coding guidelines that guide you how to structure the code of your Splunk app?

The first question defined in our quality model is about the compliance to coding guidelines. However, there are no coding guidelines available for the Splunk language which means that we have to make a first set of coding guidelines regarding the layout of SPL queries. Some of the participants mentioned that there is an existing style guide available that can automatically be applied by pressing ctrl+backslash (PC) or cmd+backslash (macOS) after clicking in the search box. According to one of the participants:

> *You should follow the Splunk SPL auto-formatting rules whenever possible. Adding additional rules is probably helpful but where you conflict with auto-formatting you are objectively wrong.*

However, this is the style guide developed by Splunk which makes it not necessarily a good style guide according to Splunk app developers. This has also been validated by one of our participants who mentioned the following:

> *The Splunk SPL auto-formatting rules are not good coding guidelines on it's own as empty lines and some indents are removed by the auto-formatting rules. However, empty lines and indents improves the readability of SPL queries.*

According to our participants, the following rules should be part of the coding guidelines that indicates how to structure a SPL query:

1. Subsearch on a new line and it should be indented
2. Every pipe should be on a new line
3. Spaces around binary operands
4. Rename should be placed at the end of the search
5. Variable names should be in snake case
6. One timechart field per line and indented when it consists of more fields
7. Descriptive fieldnames
8. Comments
9. As much selection criteria as possible (Source, sourcetype and other fields)
10. 'as' should be uppercase, 'by' as well
11. No space after [or before]
12. No eval in stats or timechart

Furthermore, the average score for each statement defined in the first block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 8. The overall average score of this block is 4.09 which corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 15.

The Splunk metric about the compliance with AppInspect and the number of comment lines scored below the average score of the entire block and below 4 which means that most of the participants did not agree with these metrics. However, this was not expected as AppInspect verifies whether the app contains a certain structure making it comply with a certain structure guideline developed by Splunk that improves the understandability of a Splunk app. One of the reasons for this relatively low score could be that some of the participants were not familiar with AppInspect which is also verified by one of the participants. This was also reflected on the scores as 4 participants answered this statement with "neutral", 11 with "agree" and 2 with "totally agree" but 0 with "disagree" or "totally disagree" as shown in Figure 15. Furthermore, one of the participants mentioned the following:

> *The compliancy with AppInspect and the deviation from coding standards are good metrics here.*

Number	Statement	Average value
1	A lower amount of deviation from the coding standard improves the compliance to coding guidelines	4.24
2	A higher amount of descriptive variable names improves the compliance to coding guidelines (The amount of descriptive variable names compared to the total number of variable names)	4.35
3	A higher amount of descriptive panel names improves the compliance to coding guidelines	4.29
4	In general, a higher amount of comment lines (up to a certain maximum) in the source code improves the compliance to coding guidelines	3.82
5	A lower amount of deviation from the "order of execution" guidelines improves the compliance to coding guidelines (order of commands in a query, order in `props.conf` etc.)	3.94
6	A lower amount of compliance errors with the Splunk app structure (using AppInspect) improves the compliance to coding guidelines	3.88
	Total average	4.09

Table 8: The average score per statement defined in the first block about the compliance to coding guidelines, by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average score of this block is 4.09.

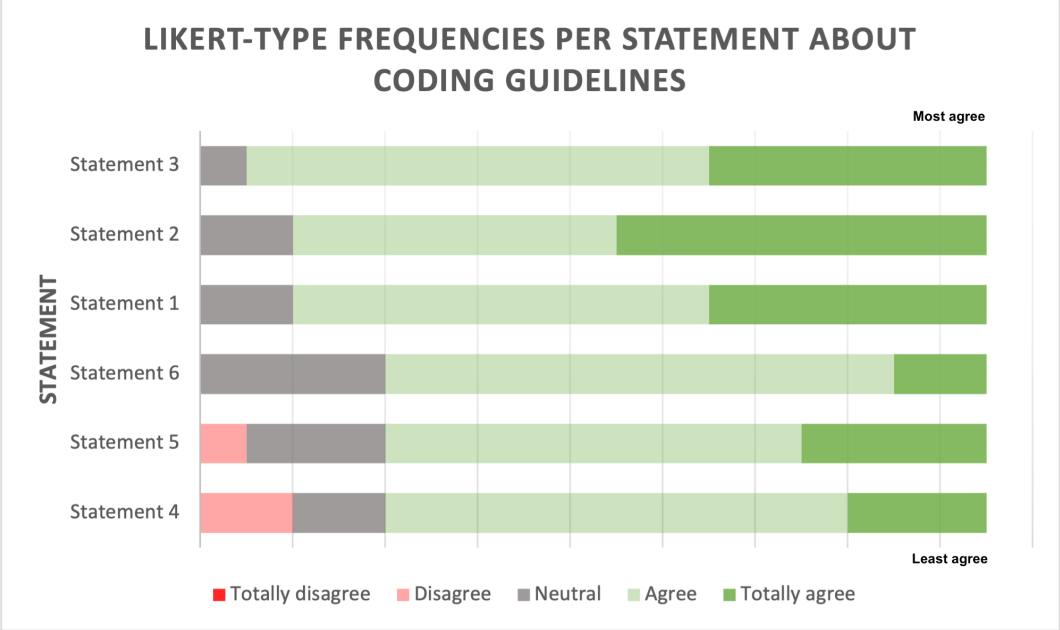


Figure 15: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 8.

7.2.1.2 What is the volume of the code?

The second question defined in our quality model is about the volume of the source code of a Splunk app. The average score for each statement defined in the second block by using the converted

Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 9. The overall average score of this block is 3.45 which corresponds to "neutral". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 16. By looking at the overall average score of the other blocks, we can see that this block has the lowest score. This is not unexpected when we look at the answers of the open-question from this block and the feedback received from our participants:

> *I don't understand this item. If you need a longer dashboard, you need it.*

According to this, some of the participants have not understood the main question of this block about the volume of the source code which has probably let to answering the statements with a low score. It is totally true that when you need a longer dashboard to show all the important information, you should implement it. However, this is not how this question should be interpreted. It should be interpreted as follows: if you need a longer dashboard to show all the important information, you should implement it, however this will cause a higher volume of the source code which makes the Splunk app less maintainable.

Furthermore, the first three statements have also a low score which can be explained by not understanding the meaning of the word "token". In the Splunk language, a token is a variable that captures and pass values in a dashboard. However, the definition of the word "token" in our survey is an instance of a sequence of characters that are grouped together as a useful semantic unit. This can also be verified by looking at the following quote mentioned by one of the participants:

> *The number of SPL lines or the number of XML lines for a dashboard (no matter what code is written).*

However, this participant answered the first three questions with "disagree" which means that the participant contradicts himself/herself or he/she didn't understand the word "token". The way we have handled this problem will be further discussed in Section 8.

In Table 9, the second statement about the number of tokens in a file has an average value of 3.24 which is the second lowest value in the table. However, when we look at Figure 16, we can see that this statement has been ranked in the top 5 of most agreed statements. This statement has not been answered with "totally disagree" by one of the participants. Furthermore, this statement has a lot of "neutral" answers which is probably caused by the problem around misunderstanding the word "token" as mentioned before.

Number	Statement	Average value
1	A lower amount of tokens in a SPL query improves the volume of the source code	3.41
2	A lower amount of tokens in a file improves the volume of the source code	3.24
3	A lower amount of tokens in a Splunk app improves the volume of the source code	3.29
4	A lower amount of files in a Splunk app improves the volume of the source code	3.41
5	A lower amount of commands in a SPL query improves the volume of the source code (E.g. a query with 2 commands is smaller than a query with 20 commands)	3.94
6	A lower amount of drilldowns on a dashboard improves the volume of the source code	3.65
7	A lower amount of views in a Splunk app improves the volume of the source code	3.53
8	A lower amount of stanzas in a `.conf` file improves the volume of the source code	3.59
9	A lower amount of Splunk knowledge objects improves the volume of the source code	3.59
10	A lower amount of fields used in a SPL query improves the volume of the source code	3.35
11	A lower amount of data sources used in a Splunk app (indexes, sources, lookups, macros, KVStores) improves the volume of the source code	3.29
12	A lower amount of dashboard tokens on a Splunk dashboard improves the volume of the source code	3.00
13	A lower amount of panels on a Splunk dashboard improves the volume of the source code	3.59
	Total average	3.45

Table 9: The average score per statement defined in the second block about the volume of the source code by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 3.45.

There was also an open-question about whether the participants think that the metrics defined in the statements give a good indication about the volume of the source code of a Splunk app and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Number of technologies used in a Splunk app (Python, java, css, html, config etc.)
2. Number of external dependencies/connections
3. Number of key-value pairs in a `.conf` file
4. Number of directories in a Splunk app

The number of programming languages used in a Splunk app is also an interesting metric to measure as literature shows that multi-language software development is useful for the developers but leads to a lower understandability and changeability of the system [MKL17].

Lastly, one of the participants indicated that it is important to make a distinction between Splunk apps and Splunk add-ons. A Splunk add-on is often used as a single component, for example

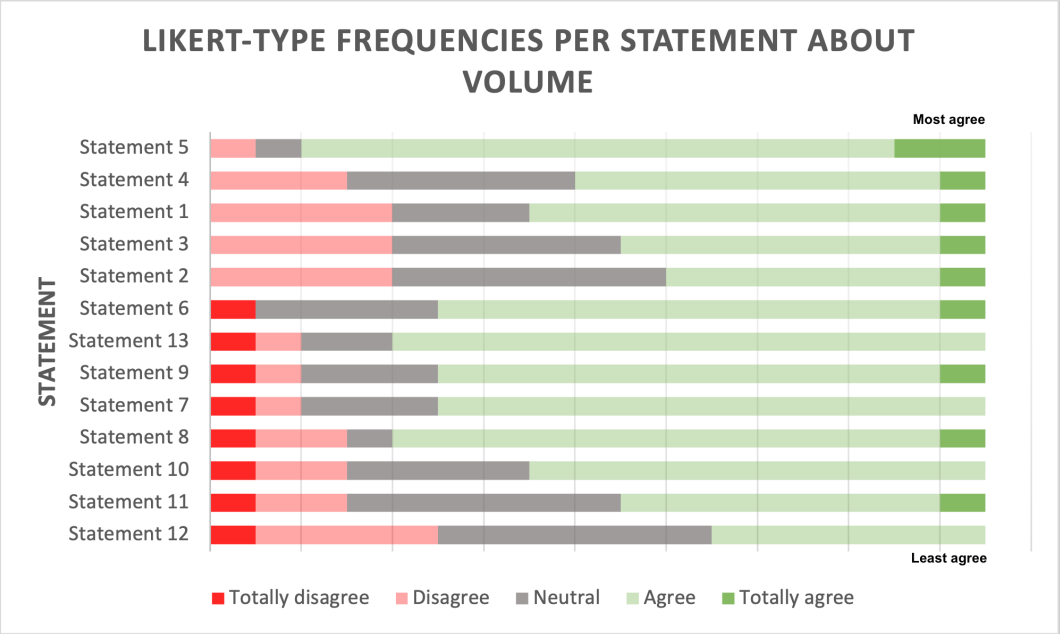


Figure 16: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 9.

to create custom commands or modular inputs, that can be re-used for several use-cases and apps. A Splunk app has a broader scope and often contains many different Splunk knowledge objects, data inputs, dashboards and in some cases one or more add-ons. Another difference between a Splunk app and add-on is that the Splunk apps are accessible by the Splunk user interface in contrast to Splunk add-ons. Furthermore, when a Splunk app also dependents on one or more add-ons, these add-ons often contains the Python code for, for example, making the connection with another platform and retrieving the desired data. Since Splunk apps can depend on add-ons, it is also important to measure the number of dependent add-ons as this affect the maintainability of a Splunk app for both the understandability, modifiability and testability.

7.2.1.3 What is the readability of the code?

The third question of our quality model is about the readability of the source code. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 10. The overall average score of this block is 3.93 which almost corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 17. The lowest score is given to the statement about the number of indentation in the source code which is not expected as the participants indicated in the first block that indentation improves the readability and should be part of the coding guidelines. Furthermore, the answers on the first statement are also not reliable for the same reason mentioned in Section 7.2.1.2 which was about a misunderstanding of the definition of the word "token".

Number	Statement	Average value
1	A lower amount of tokens in a file improves the readability of the source code (E.g. a file containing 10 tokens is easier to read than a file containing 1000 tokens)	3.76
2	A lower amount of deviation from the coding standard improves the readability of the source code)	4.35
3	A higher amount of descriptive variable names improves the readability of the source code (The amount of descriptive variable names compared to the total number of variable names)	4.65
4	A higher amount of descriptive panel names improves the readability of the source code	4.35
5	In general, a lower amount of characters on a single line improves the readability of the source code (The thresholds for the amount of characters on a single line will be calculated using a benchmark dataset)	3.59
6	A lower amount of indentation improves the readability of the source code (E.g. a query containing 0 indentation is easier to read than a query with 20 indentations, it doesn't have to be the same query)	3.12
7	A lower amount of nested searches in a SPL query improves the readability of the source code (E.g. a query containing 0 nested searches is easier to read than a query with 5 nested searches)	4.06
8	A lower amount of commands in a SPL query improves the readability of the source code (E.g. a query containing 1 command is easier to read than a query with 20 commands)	4.00
9	A lower amount of '(' characters on a single line in SPL query improves the readability of the source code	3.53
	Total average	3.93

Table 10: The average score per statement defined in the third block about the readability of the source code by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 3.93.

By using Table 10 and Figure 17, we can explain why we have chosen to use the average values in combination with the ranking based on the frequencies. In Table 10, statement 6 has been scored with an average value of 3.12 which is the lowest score in the table. In Figure 17, statement 6 has been ranked as the second lowest value instead of the lowest value according to the average value because unlike statement 9, statement 6 does not have a "totally disagree" value. However, statement 6 contains much more disagree items in comparison to statement 9 which makes us using both the table with the average scores and the figure with the ranking based on the frequencies to analyse the survey results.

There was also an open-question about whether the participants think that the metrics defined in the statements give a good indication about the readability of the source code of a Splunk app and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Number of pipes in a SPL query
2. The use of appends and joins

The number of pipes in a SPL query is also a useful metric, meaning that a higher number of pipes indicates a higher number of execution steps, resulting in a lower readability.

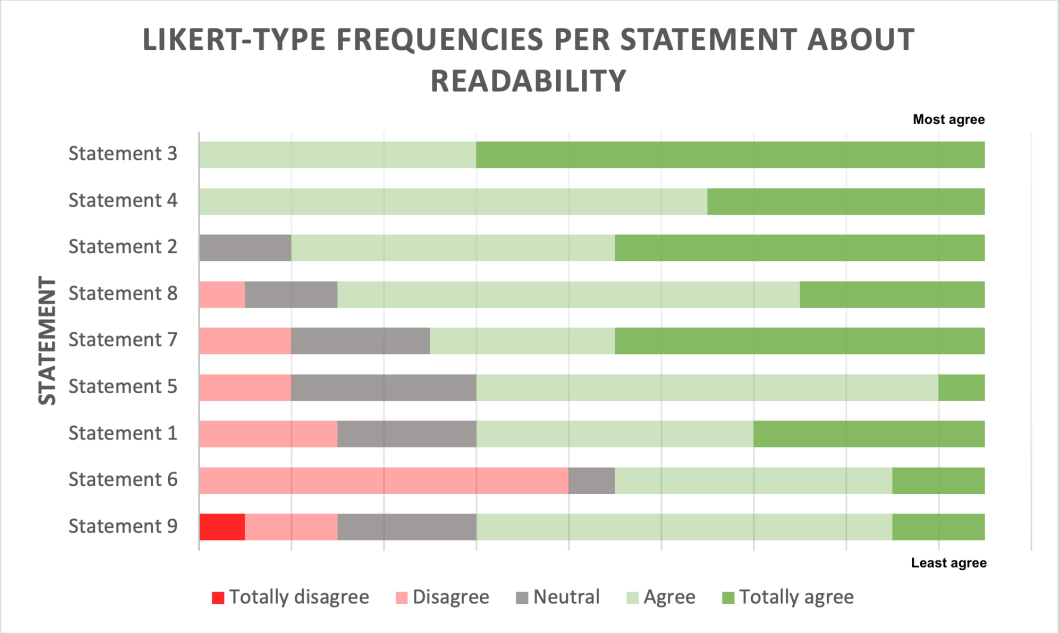


Figure 17: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 10.

7.2.1.4 Does descriptive documentation exist for the software?

The next question of our quality model is about the existence of a descriptive documentation. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 11. The overall average score of this block is 4.12 which corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 18. The lowest score is given to the statement about the amount of comments in the source code as five participants answered this statement with "neutral". This is probably because when the participant believes that a higher number of comments is not always better, he or she has answered this statement with "neutral". In Figure 18, we also see that none of the participants does not agree with this statement.

Number	Statement	Average value
1	A higher comments and identifiers consistency in a file improves the documentation of a Splunk app (E.g. are the identifiers explained in the comments)	4.24
2	A higher documentation and identifiers consistency in a file improves the documentation of a Splunk app (E.g. are the identifiers explained in the documentation)	4.06
3	A higher amount of comment lines in the source code improves the documentation of a Splunk app (E.g. source code with comment lines contains better documentation than source code without comment lines)	3.88
4	A higher degree of readability of the documentation improves the documentation of a Splunk app	4.29
	Total average	4.12

Table 11: The average score per statement defined in the fourth block about the documentation by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 4.12.

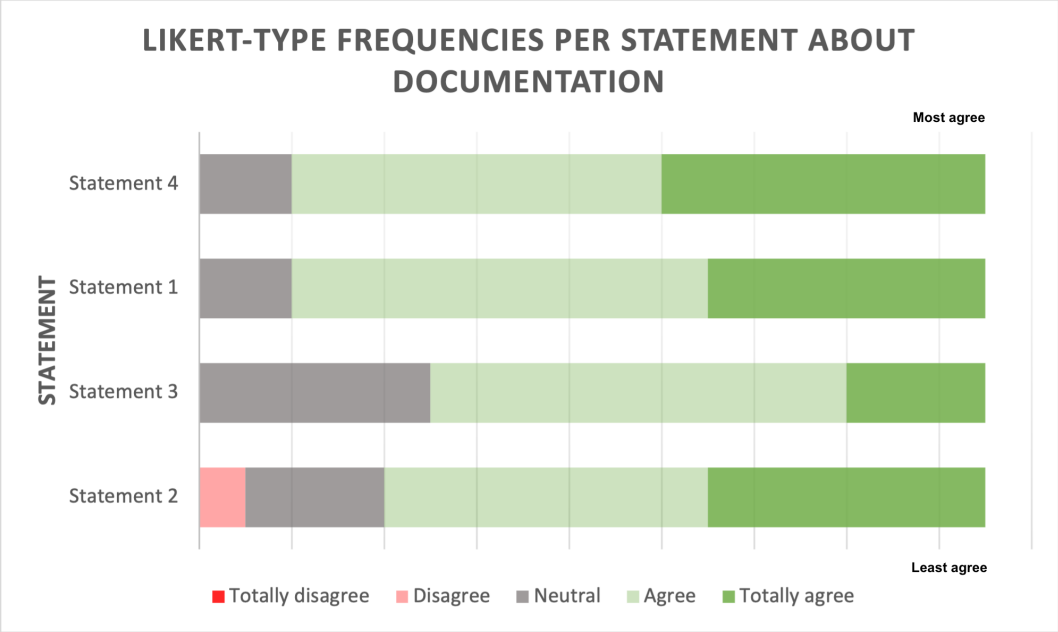


Figure 18: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 11.

Also in this block an open-question was asked about whether the participants think that the metrics defined in the statements give a good indication about whether descriptive documentation exists and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Are the data flows explained?
2. Are the sourcetypes/sources explained in the documentation?

3. Are the custom python files explained in the documentation?

It was mentioned that it is important to explain the possible data flows of a Splunk app which was also indicated during the two focus groups. Because of this, we already inserted a statement regarding the visualization of the data flows in block 10 which was given an average score of 4.24 as shown in Table 17.

7.2.1.5 Is the logging instrumentation inplace?

The next question of our quality model is about the logging instrumentation. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 12. The overall average score of this block is 3.79 which almost corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 19. The lowest score was given to the statement about the number of logging files to separate the logging. We thought that separating the logging in different groups, for example, per custom Python file would improve the usability of the logging and should therefore be included in our quality model. However, the participants indicated that, for example, having each python file log to its own log file would be overkill.

Number	Statement	Average value
1	A higher amount of logging statements (up to a certain maximum) per function improves the logging of a Splunk app	4.24
2	A higher amount of logging files to separate the logging in different groups improves the logging of a Splunk app (E.g. a logging file per custom Python file)	3.35
	Total average	3.79

Table 12: The average score per statement defined in the fifth block about the logging by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 3.79.

Also in this block an open-question was asked about whether the participants think that the metrics defined in the statements give a good indication about whether the logging instrumentation is inplace and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Number of logging statements that contains variables of that function

It is indeed useful to include variables of a specific function in the logging statements, as this gives an indication of which function is involved.

7.2.1.6 How well is the code structured?

The sixth question of our quality model is about the structure of the source code. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 13. The overall average score of this block is 3.88 which almost corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 20. The lowest score is given to the statement about the number

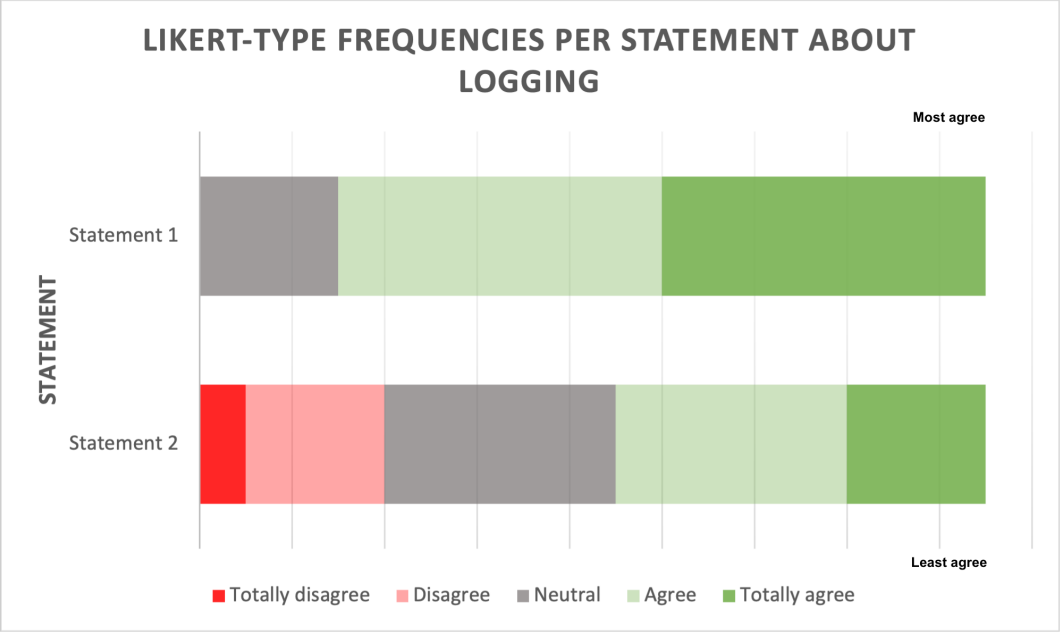


Figure 19: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 12.

of tokens in a SPL query which is probably also not reliable for the same reason mentioned in Section 7.2.1.2 which was about the misunderstanding of the definition of the word "token".

Number	Statement	Average value
1	A lower amount of tokens in a SPL query improves the structuredness of the source code	3.35
2	A lower amount of deviation from the "order of execution" standard improves the structuredness of the source code (order of commands in a query, order in `props.conf` etc.)	4.00
3	A lower amount of deviation from the coding standard improves the structuredness of the source code	4.24
4	A higher % of compliance with Splunk app structure (using Appinspect) improves the structuredness of the source code	3.94
	Total average	3.88

Table 13: The average score per statement defined in the sixth block about the structuredness of the source code by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 3.88.

Also in this block an open-question was asked about whether the participants think that the metrics defined in the statements give a good indication about the structure of the source code and if not, indicate the missing metrics. The participants indicated the following metrics:

1. A lower amount of regular expressions
2. Number of technologies used in a Splunk app (Python, java, css, html, config etc.)

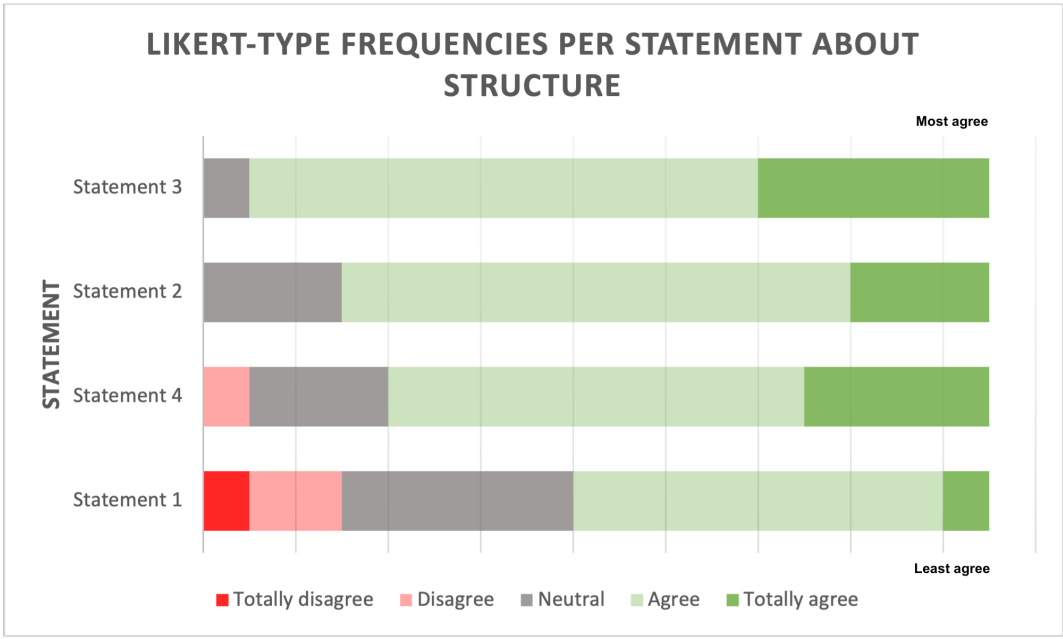


Figure 20: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 13.

As mentioned by one of the participants it is important to measure the number of regular expressions as this could indicate the number of already extracted fields where a higher percentage of fields already extracted improves the structuredness of the source code.

7.2.1.7 What is the amount of duplication in the code?

The next question is about the amount of duplication in the code. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 14. The overall average score of this block is 4.21 which corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 21. Both statements have a score above 4 and should therefore, according to the participants, be included in the quality model.

Number	Statement	Average value
1	The amount of duplication between SPL queries gives an indication about the amount of duplication in the code	4.24
2	The amount of duplication in the custom Python code gives an indication about the amount of duplication in the code	4.18
	Total average	4.21

Table 14: The average score per statement defined in the seventh block about the duplication by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 4.21.

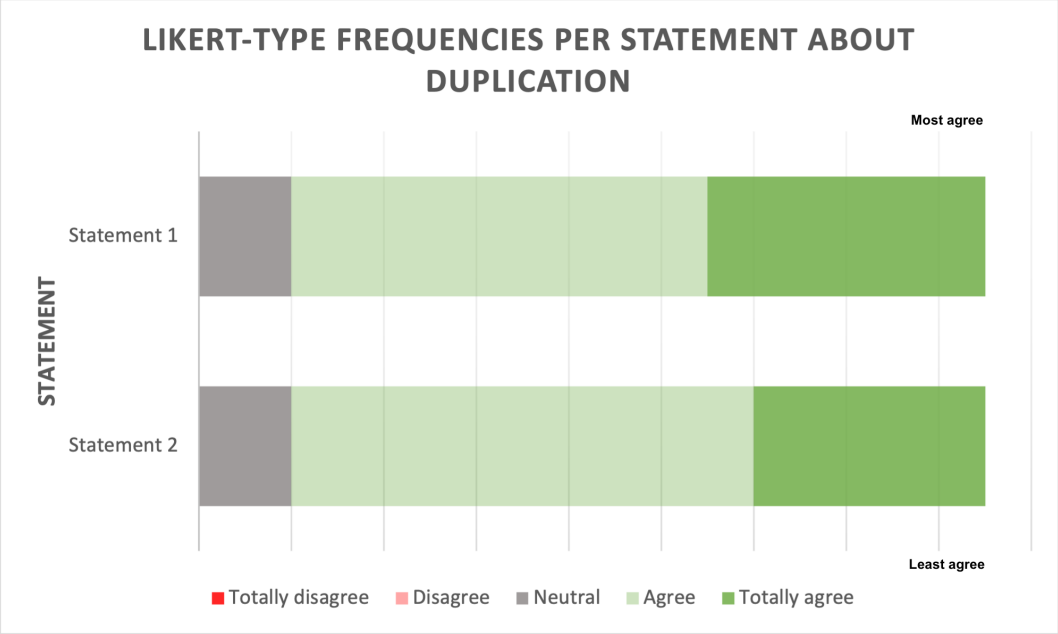


Figure 21: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 14.

7.2.1.8 What is the complexity of the source code?

The next question is about the complexity of the source code. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 15. The overall average score of this block is 3.86 which almost corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 22. The lowest score is given to the statement about the number of output data fields. However, this is not as expected as it is a factor that should be included when measuring the complexity of a SQL search [SR18]. And because the SPL language is derived from the SQL language, we expected that this metric would also be useful for measuring the complexity of a SPL query.

In Figure 22, we can see that the statement about the number of output data fields has the most "neutral" scores which could mean that the participants have not a strong opinion about whether this would increase or decrease the complexity of the app which could be the reason for this unexpected score. Furthermore, we can also see that statements three to seven have less responses than statements one and two which is caused by adding these statements after the survey was already completed by one participant.

Number	Statement	Average value
1	A lower amount of custom configuration files improves the complexity of the Splunk app (in which you need to configure the app, e.g. to enter an URL)	4.06
2	A lower amount of nested searches improves the complexity of the SPL query (E.g. a query with 0 nested searches is less complex than a query with 4 nested searches)	4.41
3	A lower amount of input data fields improves the complexity of the SPL query (E.g. a query using 1 data field is less complex than a query using 20 fields)	4.00
4	A lower amount of data tables (index, source) improves the complexity of the SPL query (E.g. a query using 1 data source is less complex than a query using 5 data sources)	3.56
5	A lower amount of sourcetypes improves the complexity of the SPL query (E.g. a query using 1 sourcetype is less complex than a query using 5 sourcetypes)	3.67
6	A lower amount of data points (lookups, KVStore, macro) improves the complexity of the SPL query (E.g. a query using only the index is less complex than a query using the index and 4 lookups)	3.81
7	A lower amount of output data fields improves the complexity of the SPL query (E.g. a query with 1 output field is less complex than a query with 20 fields)	3.50
	Total average	3.86

Table 15: The average score per statement defined in the eight block about the complexity of the source code by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 3.86.

Also in this block an open-question was asked about whether the participants think that the metrics defined in the statements give a good indication about the complexity of the source code and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Use of explicit regex commands, explicit extract commands.
2. Number of custom Python files
3. Number of different SPL commands
4. Number of key-value pairs in configuration files
5. Less sourcetypes, sources, index. However, it should be checked that the wildcard "*" is not used
6. Less custom configuration files or the availability of an interface to fill in the configuration files

The number of key-value pairs in configuration files and the use of regular expressions have been mentioned by the participants in multiple blocks and should therefore, according to the participants, be included in the quality model. The number of different SPL commands is also a useful metric where a higher number of different types of SPL commands used lead to a more complex SPL query. Another participant also mentioned that a higher number of custom configuration files indeed lead to more complexity unless there is a setup page that automatically populates the custom configuration files.

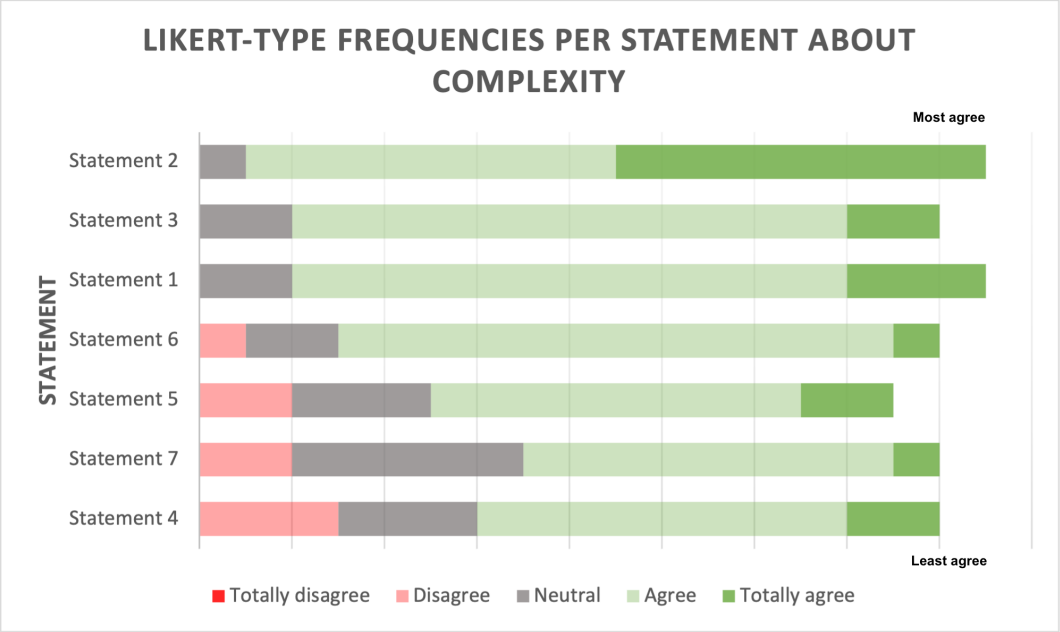


Figure 22: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 15.

7.2.1.9 What is the degree of genericity of the source code?

The ninth question of our quality model is about the genericity of the source code. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 16. The overall average score of this block is 4.19 which corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 23. All statements in this block have an average score above 4 and should therefore, according to the participants, be included in the quality model.

Number	Statement	Average value
1	A lower amount of hardcoded integers in a Python function improves the genericity of the source code (E.g. a Python function without hardcoded integers is more generic than a Python function with 10 hardcoded integers)	4.12
2	A lower amount of hardcoded strings in a Python function improves the genericity of the source code	4.18
3	A lower amount of hardcodes in a SPL query (Index, sourcetype, variables etc.) improves the genericity of the source code	4.18
4	A lower amount of duplicated hardcoded variables in a Python file improves the genericity of the source code	4.29
	Total average	4.19

Table 16: The average score per statement defined in the ninth block about the genericity of the source code by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 4.19.

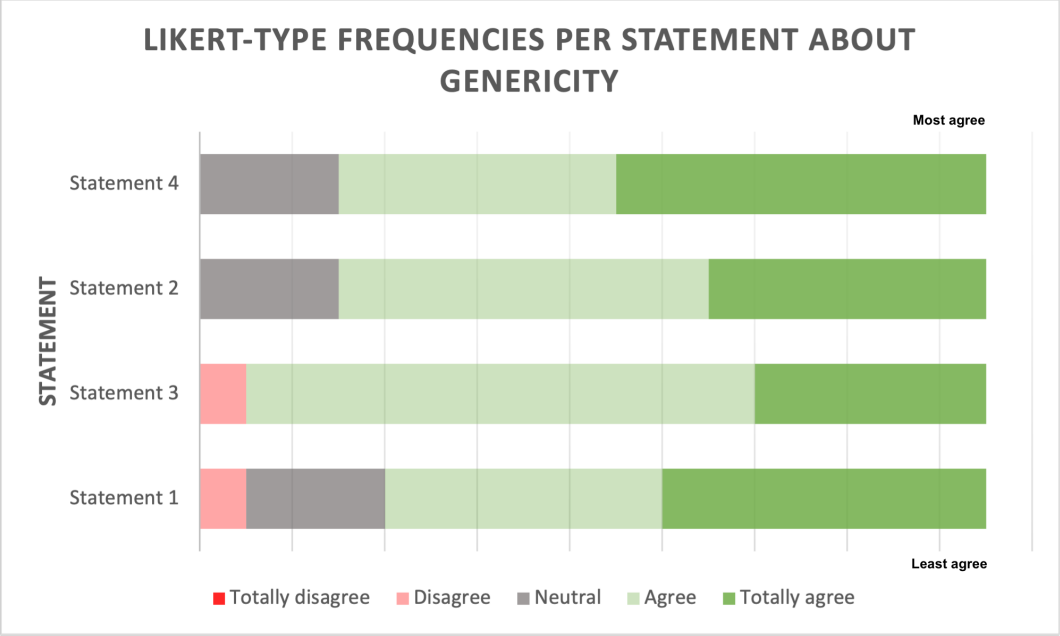


Figure 23: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 16.

Also in this block an open-question was asked about whether the participants think that the metrics defined in the statements give a good indication about the genericity of the source code and if not, indicate the missing metrics. The participants indicated the following metrics:

1. Number of macros compared to the number of searches

It was mentioned that the number of macros compared to the number of searches is an useful metric for measuring the genericity of a Splunk app where a higher number of macros compared to the number of searches improves the genericity of a Splunk app. However, it is also possible that there is one macro used in, for example, 100 SPL searches. This would result in a very low value for this metric meaning that it has a low value for genericity which is not correct. It would be better, for example, to measure the number of SPL searches containing a macro.

7.2.1.10 General questions

In the last block of the survey, we defined 10 general statements regarding the maintainability of Splunk apps which the participants can answer. The average score for each statement defined in this block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5) is shown in Table 17. The overall average score of this block is 4.18 which corresponds to "agree". In order to add more substance to these average values, we have also presented the ranking based on the frequencies of each Likert scale answer in Figure 24. All statements in this block have an average score above 4 and should therefore, according to the participants, be included in the quality model except for the statement about object-oriented programming. However, the emergence of object-oriented programming has increased the reusability,

portability, and maintainability of a system [Diw16] which makes this an unexpected outcome. One of the participants mentioned that object-oriented programming has an advantage for Python developers but Splunk app developers are not necessarily Python developers. The highest scores are given to the statements about the existence of a version control system and the existence of sample data to test the Splunk app which was also an repeated topic during the focus groups.

Number	Statement	Average value
1	The existence of a CICD pipeline improves the maintainability of a Splunk app	4.18
2	The existence of a version control system such as Bitbucket improves the maintainability of a Splunk app	4.59
3	The existence of a visualisation of the data flows improves the understandability of a Splunk app	4.24
4	The existence of unit tests improves the testability of a Splunk app	4.12
5	The existence of functional tests improves the testability of a Splunk app	4.06
6	The existence of sample data improves the testability of a Splunk app	4.53
7	The existence of a monitoring dashboard improves the testability of a Splunk app	4.06
8	A lower number of external dependencies in a Splunk app improves the testability and understandability of a Splunk app	4.29
9	The use of object-oriented programming in Python files improves the modifiability of the Python code in a Splunk app	3.65
10	The more hardcoded variables in the source code, the lower the modifiability of a Splunk app (E.G. hardcoded index in searches, hardcoded variables in the Python code)	4.12
	Total average	4.18

Table 17: The average score per statement defined in the tenth block by using the converted Likert scale answers (Totally disagree = 1, disagree = 2, neutral = 3, agree = 4, totally agree = 5). Furthermore, the overall average of this block is 4.18.

Furthermore, we also asked an open-question about how they would judge a Splunk app on its maintainability. Documentation, comments, formatting, logging, naming convention, readability, adaptability, structure and reusability are concepts mentioned by the participants. Furthermore, they indicated that it is also important to look at the number of knowledge objects and whether sample data is available. Lastly, one of the participants indicated that sorting the stanzas of a configuration file in alphabetic order improves the maintainability of a Splunk app.

Another open-question was asked about what the biggest problem is in understanding a Splunk app and why. Custom scripts, lack of sample data, lack of good documentation and lack of documentation about the data flows are concepts mentioned by the participants which are also included in our quality model. Furthermore, the participants mentioned that there is often not the possibility to debug because of, for example, to little log messages about the status. Lastly, one of the participants mentioned the following:

> *They're written very fast to get something to market immediately and then abandoned without any documentation from both Splunk and third party developers.*

This also confirms one of the problems that this master's thesis tries to address, which is that a development team often doesn't have the time to measure and visualize the maintainability and

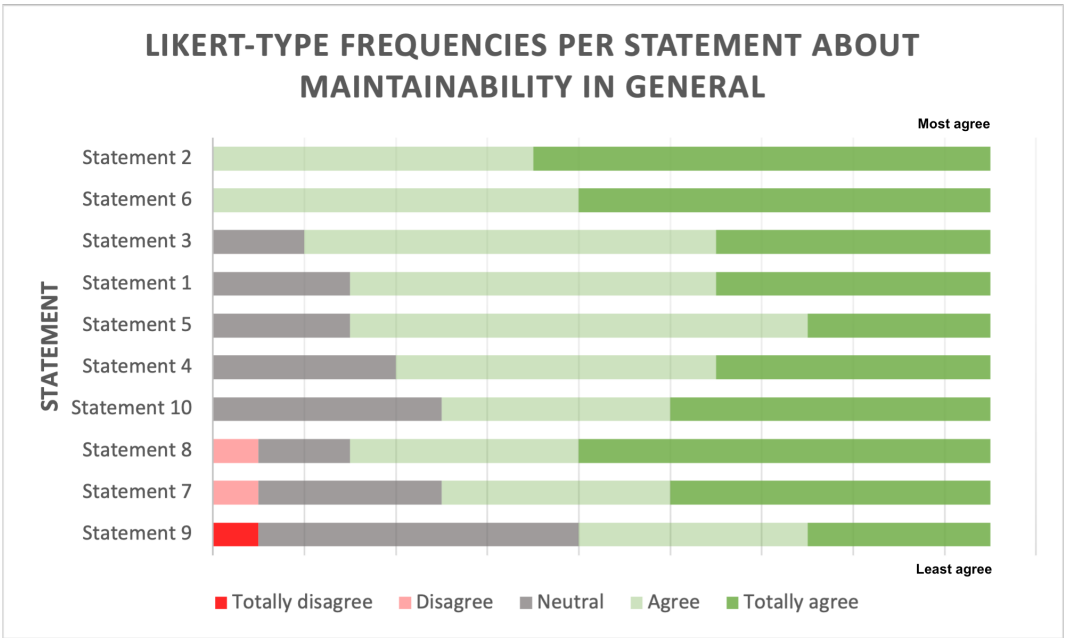


Figure 24: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 17.

wants to publish their app as soon as possible. The participants also mentioned that the cohesion between configuration files or knowledge objects is a good measure for indicating whether a Splunk app is easy to modify or expand. Furthermore, the lack of unit/functional tests makes it harder to modify or expand a Splunk app.

8 Revising the initial quality model

In this chapter, we will discuss which quality metrics we have included in our final quality model in order to measure the maintainability of Splunk apps. We will use the initial quality model as the basis which we will adjust by using the results derived from the focus groups and survey. In Section 10.2, we will verify whether the metrics give useful information and which metric is therefore a good or bad candidate for measuring the maintainability of a Splunk app. This has been done by looking at the distribution of the values of each metric.

8.1 Focus group

In the validation process of the initial quality model, we first conducted two focus groups in order to validate the goals and questions defined in our quality model. The results of these two focus groups have been presented in Chapter 7.1 which we will use here to improve the initial quality model. In this subsection, we will only discuss the parts that have been changed based on the focus group results. In Appendix B, the final version of the quality model is included.

First of all, we have changed the viewpoint of the second goal about testability by adding the consultants' viewpoint. Although the main application of our quality model is to facilitate the developers with factual data about the maintainability of a Splunk app they are developing, improving the testability of a Splunk app also ensures that consultants can install an app at the customer a lot more efficiently, saving time that can be used to deliver more value for the customer. According to our focus groups and survey, adding more value for the customer is the most important and valuable aspect within the Splunk domain. The other goals have remained unchanged as the participants of the focus groups agreed with these goals.

Furthermore, we had a discussion about the questions defined in the initial quality model. The participants mentioned that the questions defined for the first goal, about understandability, are understandable and complete except for the last question about documentation. We have changed the formulation of this goal to a "descriptive documentation" instead of a "clear documentation" as one of the participants indicated that it is impossible to have a general and objective definition of the word "clear". In general, the participants mentioned that the questions defined for the second goal, about testability, are also understandable and complete. However, some of the participants mentioned that it is important to check whether automated tests are available which we have added as a question to this goal. In general, the participants also mentioned that the questions defined for the third goal, about modifiability, are understandable and complete. However, one of the participants mentioned that it is also important to check the genericity of a Splunk app which impacts the modifiability of that Splunk app. Therefore, we have also changed the question about hardcodes to "What is the degree of genericity of the source code?".

We have also added several quality metrics to the quality model which has been indicated by the participants of the focus groups. One of the major changes we have made to our quality model is the addition of a manual checklist that measures non-automatable metrics such as the existence of an architectural overview of the possible data flows, the existence of automated tests or sample data which can be found in Table 18. Furthermore, we have added a metric for measuring the number of

external dependencies as this influences the testability of a Splunk app and a metric for measuring the number of custom configuration files as this influences the complexity of a Splunk app. There are a few more changes made based on the results of the focus groups which are included in the final version of the quality model located in the appendix.

Question	Answer (Y/N)
Is there a CICD pipeline for your Splunk app?	
Is there a version control system (e.g. Bitbucket, github) for your Splunk app?	
Is there a visualisation of the data flows for your Splunk app?	
Are there unit tests available for your Splunk app?	
Are there functional tests available for your Splunk app?	
Is there sample data available for your Splunk app?	
Is there a monitoring dashboard for your Splunk app?	
Is there a documentation available for your Splunk app?	

Table 18: A manual checklist that measures non-automatable metrics such as the existence of an architectural overview of the possible data flows.

8.2 Survey

The second validation of the initial quality model has been done by conducting a survey in order to validate the questions and metrics defined in our quality model. The results of this survey have been presented in Chapter 7.2 which we use here to improve the initial quality model. In this subsection, we will only discuss the parts that have been changed based on the survey results. In Appendix B, the final version of the quality model is included.

8.2.1 Selection criteria

As mentioned before, the goal of the survey was to validate the combination between the questions and the metrics defined in our initial quality model which results were used to improve the initial quality model. In order to create a model containing quality metrics that are indicated as important, we have removed the least important metrics indicated by the participants (currently 36 metrics in the model). This would also result in a smaller and easier to understand model that should make the usage of this model easier. In order to select the most important metrics for our third version of the quality model, we have used the following selection criteria:

1. For each question defined in our quality model, we have selected the metrics with an average score above the overall mean of that particular question with the additional condition that the average score must be above 3 (neutral).

2. After that, we have also included all metrics with an average score above 4 (agree) as this means that on average the participants agreed with this metric.

3. Lastly, we used Figure 25 that shows the ranking of each metric based on the frequencies of the Likert scale answers (totally disagree, disagree, neutral, agree, totally agree) in order to verify whether the right metrics are included in our quality model.

After we applied the selection criteria, our quality model consisted of 28 metrics instead of 36. Furthermore, we have also added additional metrics provided by the participants of the survey

which have also been assessed by us as useful which resulted in a pre-final quality model consisting of 31 metrics and a manual checklist containing 9 checks.

8.2.2 Adjustments

The adjustments based on the selection criteria mentioned above can be found in the third version of our quality model located in the appendix. For the volume question, we have included all metrics because of the misunderstanding of this question explained in Section 7.2.1.2 which makes the results unreliable. For the readability question, we also included the metric about the number of tokens in a file because of the misunderstanding of the word "token" explained in Section 7.2.1.2 which makes this result also unrealiable. For the question about the documentation, we didn't include the metric about the documentation and identifiers consistency instead we included the metric about the number of comment lines as this metric was ranked higher shown in Figure 18. None of the participants disagreed with this metric in contrary with the metric about the documentation and identifiers consistency. Furthermore, the high number of neutrals was probably caused by the problem explained in Section 7.2.1.4. For the question about the structuredness, we also included the metric about the number of tokens in a file because of the misunderstanding of the word "token" explained in Section 7.2.1.2 which makes this result also unrealiable. Except for the adjustments above, we have used the selection criteria explained in Section 8.2.1.

We have also added several quality metrics to the quality model which have been indicated by the participants of the survey as mentioned in Section 7.2. For the volume question, we have added three more metrics which are about the number of technologies used in a Splunk app, the number of key-value pairs in a `.conf` file and the number of add-ons used by a Splunk app. The last metric will be added to the manual checklist as it is not possible to automatically extract this information from the source code. There exists an `app.manifest` file containing information such as the dependent add-ons of an app, however when checking multiple Splunk apps, we found that this file was often not present. For the readability question, we didn't include the metric about the number of pipes in a SPL query indicated by one of the participants as this is highly correlated with the number of commands in a SPL query which is already part of the quality model. For the logging question, we have added the metric about the number of logging statements that contains variables of the function in which the logging statement is mentioned. For the complexity question, we have added the metric about the number of custom Python files and the number of different SPL commands.

9 Creating a Splunk app

In this chapter, we will discuss the difference between the vision of this thesis and the current implementation. Furthermore, we will show a design of how the tool should look like in the end.

9.1 Difference between the vision and the current implementation

The vision of this thesis is to create a tool, which is a Splunk app, containing all measurements explained in Chapter 8. The desired tool consists of three elements which is a version control system (e.g. Bitbucket), Python code for the measurements, and Splunk for visualizing the retrieved information as can be seen in Figure 26.

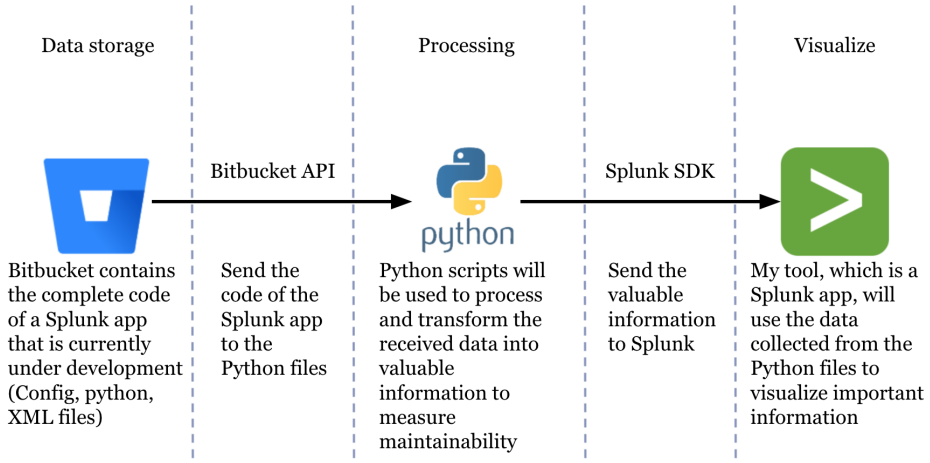


Figure 26: The desired tool consists of three elements which are a version control system (e.g. Bitbucket), Python code for the measurements, and Splunk for visualizing the retrieved information.

We have also created an architectural overview containing the data flows of the desired tool in more detail. The architectural overview has been created by making a concept using the knowledge and experience of the thesis author as a Splunk developer. We have validated this concept by illustrating and explaining the concept to a Splunk expert. The feedback of the Splunk expert has been implemented in our concept resulting in a final version of the architectural overview which is shown in Figure 27. First of all, the source code of a Splunk app should be stored in a version control system such as Bitbucket. After each pull request, which means that the source code has been changed, the entire code (only first time) and changes will be sent via the HEC (HTTP Event Collector) to a Splunk index. In Bitbucket this can be achieved by creating a Bitbucket pipeline that will be triggered when a pull-request has been initiated. Secondly, an alert action should be created that will be triggered when new data is received by the indexer. This will cause the Python scripts to be executed which calculate the defined quality metrics and send the results to Splunk via the Splunk SDK.

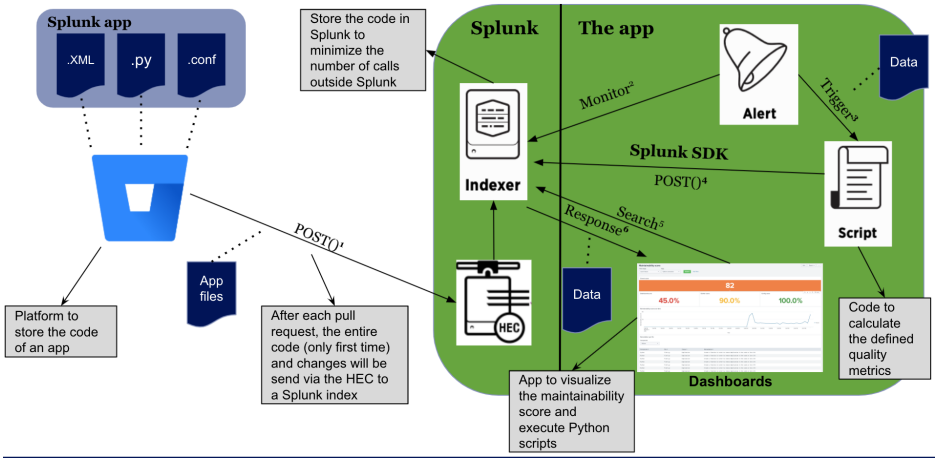


Figure 27: The final version of the architectural overview for the desired tool. (1) The source code of a Splunk app should be stored in a version control system such as Bitbucket. After each pull request, the entire code (only first time) and changes will be sent via the HEC to a Splunk index; (2/3/4) An alert action should be created that will be triggered when new data is received by the indexer and causes the Python scripts to be executed; (5/6) The Splunk app contains two dashboards to visualize the maintainability score of the entire source code of a particular app and of the source code that has been changed.

Lastly, the Splunk app contains two dashboards (1) to visualize the maintainability score of the entire source code of a particular app; (2) to visualize the maintainability score of the source code that has been changed according to the pull-request. The design of these two dashboards can be found in Figure 28.

However, because of time constraints and the size of this thesis, we will not implement the desired tool as explained above. We will implement the tool as follows:

1. We will not implement all quality metrics defined in the final version of the quality model. We have chosen to implement the Splunk metrics defined for the volume question in our final version of the quality model. For these metrics, we have implemented the code and made the benchmark rating system as explained in Section 2.4. By doing this, we are able to show the feasibility of this project.

2. We will not make a connection with a version control system such as Bitbucket. We have chosen to develop the tool such that it retrieves the source code of a particular app by searching in the `app` directory of Splunk.

3. We will not create an alert action that will be triggered when new data is retrieved by the indexer. We have chosen to create an alert action that will be triggered every 1 minute in order to demonstrate our tool.

4. Lastly, because we will not make a connection with a version control system, we will also not create a dashboard that visualizes the maintainability score of the source code that has been changed according to a pull-request.

As a result of the difference between the vision and the current implementation, we will also only define the thresholds and create the benchmark rating system for the Splunk metrics defined for the volume question. The explanation in Chapter 2 and the illustration in Chapter 10.2, should provide enough information to define the thresholds and create the benchmark rating system for the other metrics defined in the final quality model.

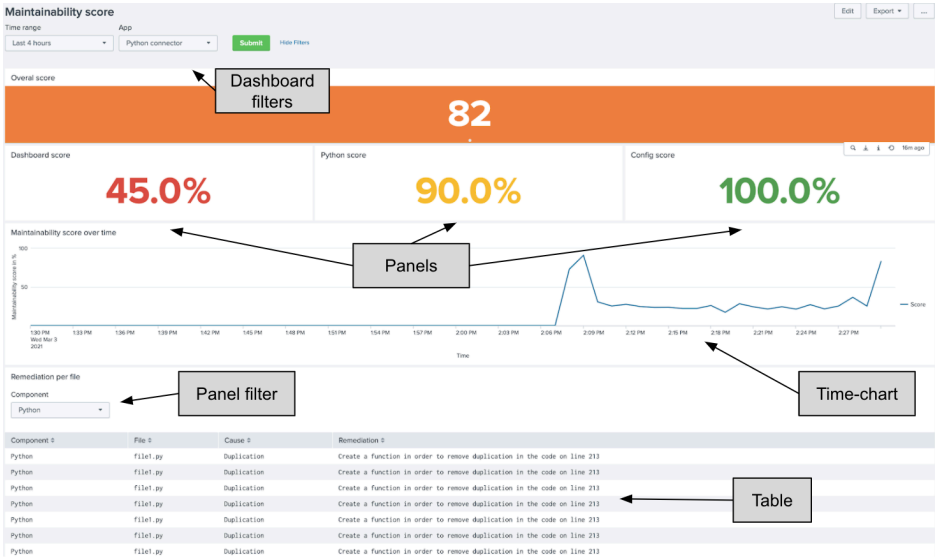


Figure 28: A design of the dashboard to visualize the maintainability score of the entire source code of a particular app.

9.2 Current implementation

As mentioned in previous section, we have only implemented the quality metrics defined for the volume question in our quality model. The volume question in our quality model consists of 15 quality metrics which is thereby the largest question of our quality model. For each metric, we have implemented a custom Python function that returns the metric and weight value for a given Splunk app. These Python functions are implemented in a custom Python file that serves as a library file which can be used by other Python files. This custom Python file currently consists of three classes which are a class for the logging, a class for the volume question and a class for additional data processing. It should, eventually, consists of a class for each question in our quality model with the needed metric calculations. We have also implemented a second custom Python file that calls the functions in the library file for a given Splunk app. This is also the file that should be executed when the alert action is triggered.

The following two metrics, number of fields used in a SPL search and number of data sources used in a SPL search, are also defined for the volume question in our quality model. However, we have not yet implemented these two metrics for the current implementation because of the difficulties of these two quality metrics and the time needed to implement these metrics. We have chosen to add the implementation of these metrics to the future work, because of the aforementioned time-constraints.

10 Creating a rating system for each metric based on a benchmark set

In this chapter, we will discuss how the method for creating a benchmark rating system has been applied within this thesis. Furthermore, we will explain the benchmark dataset and illustrate how the benchmark rating systems have been created.

10.1 Benchmark dataset

As mentioned in previous sections, we will focus on Splunk apps instead of Splunk add-ons which means that our benchmark dataset should consist of the source code of multiple Splunk apps. Unfortunately, Github doesn't have the possibility to search for and download Splunk apps which makes it not possible to use the Github API. Because of this, we have to use Splunkbase and download the Splunk apps manually. We have only added the source code of a Splunk app to our benchmark dataset when the app contains a `default` directory with `.conf` and `.xml` files. This will not lead to a selection bias because Splunk defines a Splunk app as an app containing many different Splunk knowledge objects, data inputs, dashboards and in some cases one or more add-ons. Furthermore, it is required to have a `default` folder containing the `app.conf` and `default.xml` file. By adding this requirement, we ensure that only Splunk apps as defined by Splunk are included and that all Splunk metrics defined for the volume question can be measured for each Splunk app in our benchmark dataset. This has resulted in a benchmark dataset containing the source code of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2).

This dataset consists of a small fraction of the 700 available Splunk apps located on Splunkbase. The reason for this is that we have to check each Splunk app manually on the presence of `.conf` and `.xml` files. Furthermore, we have to review each app and remove third-party code manually which is a time consuming task. We have done this because the third-party code does not fall within the maintenance scope and should therefore not be included in the maintainability score of that Splunk app. Examples of third-party libraries are: `yaml`, `websocket`, `weaviate`, `splunklib`, `nltk`, `dockerlib` and more. The size of the Splunk apps in our benchmark dataset ranged from 2503 to 14,619,072 tokens in total.

10.2 Benchmark rating system

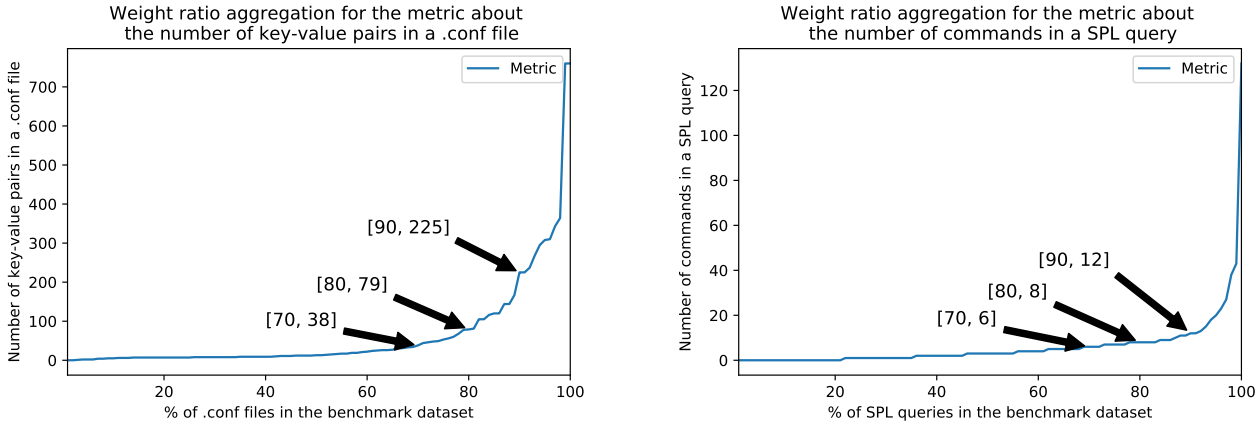
For each app in our benchmark dataset, we have measured the needed values, e.g. metric and weight value, for each metric defined for the volume question in our quality model. We have done this by implementing Python scripts that automatically measure these needed values and saves the values per metric in a separate `csv` file which are used in the next subsections.

10.2.1 Thresholds

As mentioned before, we will only create the benchmark rating systems for the quality metrics defined for the volume question in our quality model. We will use the method described in Section 2.4, however, for one of the metrics, we have slightly deviated from the method as it is explained in Section 2.4. This metric is about the number of knowledge objects in a Splunk app. Here, it is

not possible to define a weight value as we can't derive this value in the form of "the number of tokens of the entity". For this metric, we have calculated the frequency of each metric value and for each metric, using the frequency, the percentage of the apps in our benchmark dataset having that metric value. These percentages are then used for the fifth and sixth step in the method as it is explained in Section 2.4.

For the thresholds, we have first used the 70/80/90% values to split the data into low ($0-70\%$), moderate ($70-80\%$), high ($80-90\%$) and very-high risk ($>90\%$) which is also proposed by Alves et al. [AYV10]. It is also possible to adjust the percentages for the thresholds for a particular metric, for example, when the values are very close to each other [vdB16]. In Figure 29, we have showed two examples of the metric distributions which are used to define the threshold values.



(a) Number of key-value pairs in a `.conf` file.

(b) Number of commands in a SPL query.

Figure 29: The metric distributions for two metrics for which a benchmark rating system will be created.

When looking at these distributions, it seems that there is a good distribution and no need to adjust the percentages for the thresholds. However, by looking more closely to the underlying data, we can see in Figure 30 that 29 Splunk apps in our benchmark dataset have only metric values below the low threshold which is almost 60% of the benchmark dataset. Furthermore, when we look even deeper into the underlying data, we can see in Figure 31 that only a few apps (5,7,10,33,37) cause that the 90% threshold has such a high metric value. This issue would probably disappear when using a larger benchmark dataset, because with a benchmark dataset of 50 Splunk apps, one Splunk app could, in worst case scenario, contribute for a maximum of 2% to the entire benchmark (because we have normalized the weights [AYV10]). For example, when a Splunk app in our benchmark dataset contains only one entity with a very large metric value, this metric value would contribute 2% to the benchmark. This means that in worse case scenario only 5 Splunk apps would be part of the $90-100\%$, 5 of the $80-90\%$, 5 of the $70-80\%$ and 35 of the $0-70\%$. This would cause that it is not possible to define 4 different thresholds to fit a $5{-}30{-}30{-}30{-}5\%$ distribution for the rating system discussed in the next section.

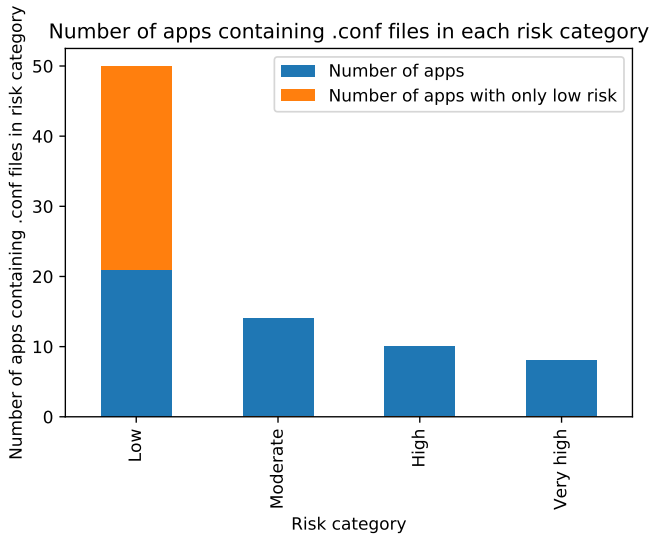


Figure 30: The risk distribution for the metric about the number of key-value pairs in a `.conf` file. Here, we can see that all Splunk apps in the benchmark dataset contain `.conf` files in the low risk category. However, a total of 29 Splunk apps (orange) have only `.conf` files in the low risk category and none in the other risk categories.

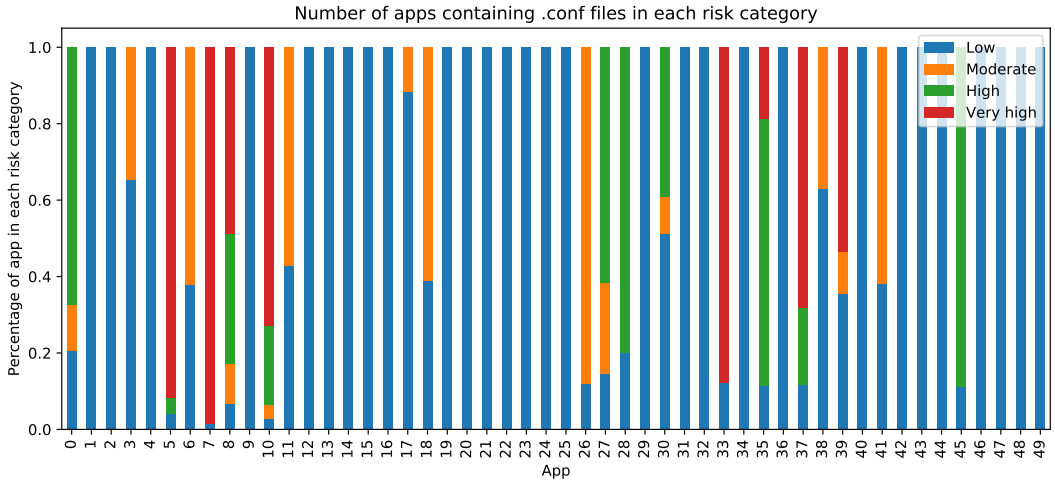


Figure 31: A more detailed overview of the risk distribution per app for the metric about the number of key-value pairs in a `.conf` file. Here, we can see that only a few apps (5,7,10,33,37) have caused the high metric value for the 90% threshold.

Because of the aforementioned problem, we have chosen to adjust the percentages of the thresholds from 70/80/90% to 40/60/80% for the following metrics:

1. Number of tokens in a file
2. Number of dashboard tokens on a dashboard
3. Number of drilldowns on a dashboard

4. Number of stanzas in a `.conf` file
5. Number of panels on a dashboard
6. Number of key-value pairs in a `.conf` file

This ensures that a better risk distribution per app exists as can be seen in Figure 32 which makes it possible to define 4 different thresholds to fit a 5−30−30−30−5% distribution for the rating systems discussed in the next section.

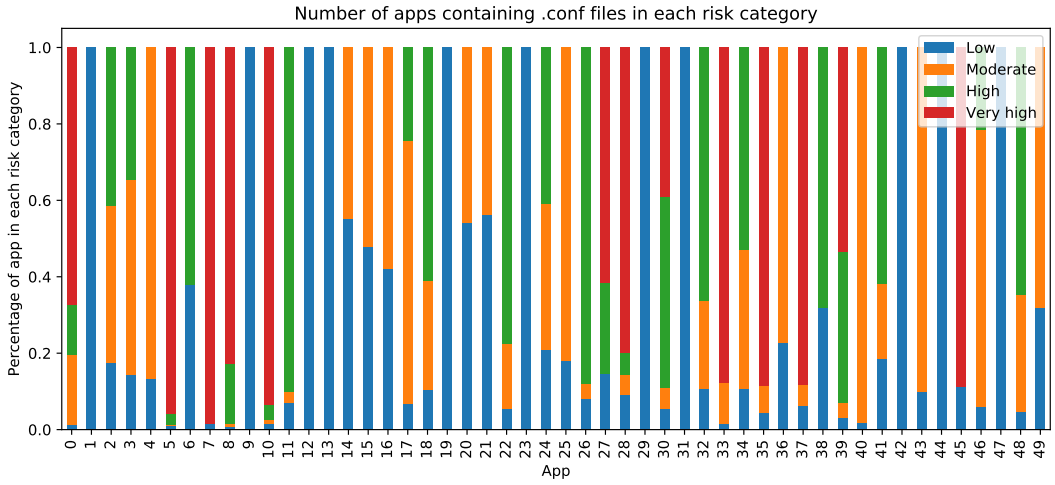


Figure 32: A detailed overview of the risk distribution per app for the metric about the number of key-value pairs in a `.conf` file by using the 40/60/80% values. Here, we can see that a better risk distribution per app exists which makes it possible to define 4 different thresholds to fit a 5−30−30−30−5% distribution for the rating systems discussed in the next section.

For the other metrics defined for the volume question in our quality model, we have also defined the thresholds as explained above by using the 70/80/90% values as proposed by Alves et al. [AYV10]. In Table 19 and 20, we have shown how the metric thresholds are defined by using the metric distributions as shown in Figure 29. The metric distributions and metric thresholds for the other metrics can be found in Appendix D. These thresholds are based on the benchmark dataset used in this thesis containing the source code of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2). Therefore, these metric thresholds may change when using a benchmark dataset with the source code of more or other Splunk apps. Lastly, we have removed the metric about the number of drilldowns on a dashboard from our quality model as the metric values were not as useful as we thought. In Appendix D, we can see that the metric value is 0 for more than 70% of the total number of tokens in the dashboard files of our benchmark dataset. It is also not useful to increase the threshold percentages to 80/90/95% which is done by Alvez et al. [AYV10] because this makes it not possible to fit a 5−30−30−30−5% distribution for the rating systems discussed in the next section.

Risk category	Number of key-value pairs
Low	0 - 9
Moderate	10 - 22
High	23 - 79
Very high	80+

Table 19: The metric thresholds for the metric about the number of key-value pairs in a `.conf` file.

Risk category	Number of commands
Low	0 - 6
Moderate	7 - 8
High	9 - 12
Very high	13+

Table 20: The metric thresholds for the metric about the number of commands in a SPL query.

10.2.2 Rating system

After defining the metric thresholds, we have created the benchmark rating systems using the two steps explained in Section 2.4. First of all, we have done the first level aggregation in which risk profiles per metric per Splunk app are created using the thresholds defined in previous section. After that, we have calculated the cumulative risk profiles per Splunk app per metric which is also explained in Section 2.4. This has resulted in 50 cumulative risk profiles per metric because our benchmark dataset consists of 50 Splunk apps. An example of how the cumulative risk profiles looks like by using the risk profiles of the metric about the number of tokens in a SPL query can be found in the appendix C.

The second step is the second level aggregation in which the rating thresholds are defined that indicates the maximum cumulative percentage per risk category (moderate, high and very high). As mentioned by Alves et al. [ACV11], we have to set two parameters which are the number of ratings (stars) and the desired distribution.

1. Number of ratings: we want a 5-star rating as this is also used by Alves et al. [ACV11]. Furthermore, this type of rating is used in many other research, for example, the use of the 5-point Likert scale in surveys or the rating of movies.

2. Desired distribution: we will define the thresholds such that the data fits a 5−30−30−30−5% distribution as this is also used by [vdB16][BCSV12]. This would result in the best 5% of our benchmark dataset receiving 5 stars, the next 30% would receive 4 stars and so on. By using this distribution, we could identify the 5% worst and best systems in our benchmark dataset.

These rating thresholds are based on the benchmark dataset and used to indicate the number of stars a new Splunk app would receive. The cumulative percentages of each risk category of this new Splunk app should be equal or lower than the thresholds for rating 3 and higher than the thresholds for rating 4 to receive a 3 star rating. An example of the initial and optimized rating thresholds can be found in Table 21 and 22 which have been calculated using the algorithm proposed by Alvez et al. [ACV11]. For the other metrics defined for the volume question in our quality model, we have also calculated the rating thresholds as explained above. The optimized rating thresholds for these metrics can be found in Appendix E. In this master thesis, we will use the average ratings of all metrics in our quality model as the maintainability score.

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	13.2%	0%	0%
***	35.7%	17.0%	0%
**	66.5%	54.8%	28.8%
*	–	–	–

Table 21: The initial rating thresholds for the metric about the number of tokens in a SPL query.

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	13.2%	0%	0%
***	35.7%	24.0%	0%
**	70.2%	54.8%	32.0%
*	–	–	–

Table 22: The optimized rating thresholds for the metric about the number of tokens in a SPL query.

For some of the metrics, it is not useful to create a benchmark rating system based on the method explained above. Here, it is about the quality metrics having only one value per Splunk app such as the number of tokens in a Splunk app, the number of files in a Splunk app or the number of technologies used in a Splunk app. The reason for this is because these metrics have the same distribution as the percentages used in the metric threshold derivation which is 70/80/90%. This means that there are, in our benchmark dataset of 50 Splunk apps, 5 Splunk apps with the cummulative percentages of 1-1-1 (90-100%), 5 Splunk apps with 1-1-0 (80-90%), 5 Splunk apps with 1-0-0 (70-80%) and 35 Splunk apps with 0-0-0 (0-70%). Therefore, a distribution of $5-30-30-30-5\%$ for our rating system would result in a rating system with almost all similar thresholds as can be seen in Table 23.

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	0%	0%	0%
***	0%	0%	0%
**	100%	100%	100%
*	–	–	–

Table 23: The optimized rating thresholds for the metric about the number of technologies used in a Splunk app.

Because of this, we have chosen to use only one calibration level instead of the 1st and 2nd level calibration explained by Alves et al. [ACV11]. This means that the metric values for the apps are directly mapped to rating thresholds without defining the metric thresholds for low, moderate, high and very high risk as explained in Section 10.2.1. In order to define the rating thresholds such that the data fits a $5-30-30-30-5\%$ distribution, we will use the same algorithm but without the risk categorization which can be seen in Table 24. This approach has also been used by Heitlager et al. [HKV07] but in this master thesis the rating thresholds are based on a benchmark dataset instead of using expert opinions. By using this approach, it is still not possible to define rating thresholds such that the data fits a $5-30-30-30-5\%$ distribution for the metric about the number of knowledge objects in a Splunk app as can be seen in Table 55. This is caused by the fact that there are 24 Splunk apps in our benchmark dataset without any knowledge object resulting in a metric value of 0. However, the participants of the focus groups and survey indicated that this metric is very important regarding maintainability and should therefore be included in the model.

Because of this, it would be interesting to investigate how the metric values would be distributed when, for example, using a larger dataset.

Rating	**Number of tokens**
*****	$0 - 5439$
****	$5439 - 16.990$
***	$16.990 - 97.228$
**	$97.228 - 3.008.490$
*	$> 3.008.490$

Table 24: The optimized rating thresholds for the metric about the total number of tokens in a Splunk app.

11 Tool validation

In this section, we will discuss how we have validated our quality model and prototype of our tool for automatically measuring and monitoring the maintainability score of a Splunk app. For this validation step, we have chosen to conduct three interviews with employees of SMT. We have chosen for this because, first of all, we do this master thesis in collaboration with SMT which makes it important to receive feedback from them about the usability of this model. Secondly, because of time-constraints we were not able to conduct a survey among Splunk developers inside and outside SMT. Last but not least, we will only explain how the metric and rating thresholds are derived, how this will be used within the tool and how the tool should look like in the end. Because of this, interviews are more useful in gathering relevant feedback for this validation step.

1. First of all, we have presented the final quality model.
2. Secondly, we explained how the metric and rating thresholds are calculated and how this will be used within the tool.
3. Thirdly, we have presented a prototype of the tool.
4. Lastly, we asked the following two questions:
   (a) Do you think this model/tool is useful in practice?
   (b) Should we make adjustments to the model/tool?

11.1 Results

In general, the three interviewees were very enthusiastic about the created model and desired tool. However, during the interviews they have shared interesting thoughts about the model and tool:

Do you think this model/tool is useful in practice?
The interviewees indicated that the model was indeed useful in practice and one of the participants also mentioned a constraint of the usability of this model which was the ability to automate the process. If it is not possible to automate the entire process, it will cost too much time which will probably lead to not using this model/tool. Furthermore, the participants also mentioned that it is very important to validate the model/tool in practice. Because, it is possible that a certain Splunk app is ranked with a very high score, however, does this also mean that the Splunk app is indeed better to maintain in practice. For example, it would be very interesting to compare the 5% worst and 5% best Splunk apps of our benchmark dataset in practice. Is the time, to solve a bug in the 5% worst Splunk apps, indeed much longer than the time needed to solve a bug in the 5% best Splunk apps. When this is validated, we could give our clients advice whether or not to install a specific app based on the maintainability score which will, eventually, lead to time and costs savings. Furthermore, one of the participants indicated that it should be possible to adjust the metric thresholds for certain metrics in order to be useful in practice.

Should we make adjustments to the model/tool?
One of the participants mentioned that the model/tool, itself, is complete and should not be

adjusted. However, he mentioned that it would be very interesting to add a new functionality to the tool and the model that it embodies which probably lead to a better business case. Instead of only using this tool during the development of a Splunk app, it would be interesting to use this tool for rating already existing Splunk apps on, for example, Splunkbase. It often happens that there are 5 or 6 Splunk apps available on Splunkbase with more or less the same functionalities which makes it very difficult to choose which Splunk app is best to install. However, when it is possible to use this tool to rank all 5 or 6 Splunk apps based on the maintainability, we can use this score to indicate which Splunk app is probably best to install. For example, at SMT we are responsible for the maintenance of Splunk apps that are installed at our clients and to solve issues that occur. Therefore, it is important to install apps at our clients that are easy to maintain which, eventually, saves us a lot of time and effort.

Furthermore, one of the participants indicated that the calculation for the overall maintainability score should be adjusted. In this master thesis, the maintainability score is defined as the average ratings of all metrics in our quality model. However, he mentioned that this approach is sensitive to outliers and should therefore be adjusted. He suggested to use the median instead of the mean because the median is less sensitive to outliers. Furthermore, he mentioned that the outliers should be clearly visualized on the dashboard of the tool. However, when using the median with numbers ranging from 1 to 5 it could lead to a distorted picture of the maintainability score because a Splunk app with a 5-star rating for most of the metrics and a few outliers with a 1-star rating would result in an overall maintainability score of 5. This would mean that the Splunk app has the highest rating and should therefore not be adjusted without knowing that there are some parts of the Splunk app with a problematic rating.

12 Discussion

In this section, we will discuss the results and method used in this master thesis. We will discuss the small sample size and benchmark dataset. Furthermore, we will discuss the difference between apps and add-ons, the selection criteria we have used to select the quality metrics for our quality model and the lack of real-world validation. Lastly, we will discuss how the results can be generalized to other big data platforms and some aspects for further development.

12.1 Small sample size

In this master thesis, we have used a combination of quantitative (survey) and qualitative (focus group) research methods to validate our initial quality model. First of all, we have conducted two focus groups with 4 participants in both sessions. This is a limitation of my qualitative research as research has shown that, preferably, you should do 4 to 6 focus groups consisting of 3 to 12 participants [KBL08]. However, for the purpose of this master project, these two focus groups have provided us with valuable information that can be used to improve the initial quality model. Secondly, we have conducted a survey with 17 participants in total which is also a limitation. There are a number of factors that could have caused this relative small amount of participants (17). First of all, the segment for our participants is narrowed to only Splunk app developers. Secondly, after reaching out to possible participants, they must be willing to spend between 20 to 30 minutes for filling in the entire survey. However, besides the low number of participants of the survey, they have provided us with valuable and reliable data as the average time spent completing the survey is around 1 hour and 22 minutes. Furthermore, they also left valuable information in the open questions which has let to the addition of new metrics or the deletion of old metrics.

12.2 Benchmark dataset

In order to give a Splunk app a maintainability score that is backed by data, we have created a benchmark rating system for each metric defined for the volume question in our quality model. However, the benchmark dataset contains the source code of only 50 Splunk apps which is a small fraction of the total available Splunk apps (700) on Splunkbase. It could be the case that these Splunk apps have a certain bias which could cause that our benchmark rating systems do not reflect the maintainability score well. This could, for example, happen when the benchmark dataset only contains very good Splunk apps which cause that our rating system will give less maintainable Splunk apps a very low rating and vice versa. We have also selected the Splunk apps for our benchmark dataset by repeatedly choosing a random app on Splunkbase. However, this could cause a certain bias which could be mitigated by using an algorithm to select the Splunk apps randomly instead of a human.

Furthermore, we have replaced the source code of two Splunk apps from our randomly selected benchmark dataset. These two Splunk apps contain Javascript files with a size of over 20MB per file which take too much time for measuring the metric values such as the number of tokens in a file. The measurement of the number of tokens in one of the Javascript files, which was only 10MB instead of 27.6MB of another file, took approximately 4.5 hours. For measuring the number of tokens in a file, we are using the `word_tokenize` library, however, for these large Javascript files it

takes to much time to calculate using a MacBook Pro. Because of time-constraints, we are not able to execute the scripts for these two large Splunk apps and are therefore replaced by the source code of two other Splunk apps.

Also, some of the Splunk apps in our benchmark dataset contain very large CSV files which caused that some metrics have large metric values. In some Splunk apps, the CSV files were used as sample data which is probably not subjected to maintainability and in other Splunk apps it was used to enrich the data in Splunk which is subjected to maintainability. However, we were not able to judge whether a CSV file was subjected to maintainability for each app and have therefore not removed these files from the benchmark dataset. It would be interesting to check, with the help of an expert, which CSV files are subjected to maintainability and what the effect is on the metric and rating thresholds when removing the CSV files that are not subjected to maintainability.

Lastly, some of the Splunk apps in our benchmark dataset contain unreadable files which are therefore not used for the calculation of the metrics. Examples of such files are `.gif` and `.woff` files. It would be interesting to investigate what the impact of such files is on the overall maintainability of a Splunk app.

12.3 Difference between apps and add-ons

In the survey, one of the participants indicated that it is important to make a distinction between Splunk apps and Splunk add-ons. A Splunk add-on is often used as a single component, for example to create custom commands or modular inputs, that can be re-used for several use-cases and apps. A Splunk app has a broader scope and often contains many different Splunk knowledge objects, data inputs, dashboards and in some cases one or more add-ons. When a Splunk app also depends on one or more add-ons, these add-ons often contains the Python code for, for example, making the connection with another platform and retrieving the desired data. Since Splunk apps can depend on add-ons, it is also important to take the maintainability of these add-ons into account. In some cases, the app requires an add-on to be installed where an add-on can often be used with or without a Splunk app. However, this is not taken into account in this thesis, we have only defined a quality metric that measures the number of dependent Splunk add-ons. The reason for this is that it would take a lot of time to discover which Splunk add-ons are required or have a dependency with a Splunk app for each Splunk app in our benchmark dataset.

Furthermore, it is important to mention that the metric and rating thresholds are defined by using a benchmark dataset containing the source code of 50 Splunk apps. Therefore, these metrics and values can not be used for rating a Splunk add-on as it is built for a Splunk app.

12.4 Selection criteria

In order to select the final set of quality metrics for our quality model, we have defined specific selection criteria that have been used on the analysed survey results. First of all, we have used the average score of each metric to select the initial set of quality metrics. However, because our survey consists of stand-alone Likert questions which are part of the ordinal measurement scale, it is not recommended to use averages [BB12]. Therefore, we have also used the frequencies of

each Likert scale answer (totally disagree, disagree, neutral, agree, totally agree) in order to add more substance to the average scores. These frequencies have been used to verify whether the right metrics are included in the initial set of quality metrics based on the average scores.

The defined selection criteria has resulted in a final quality model containing quality metrics that were expected to be included. However, the deletion of the metrics about whether object-oriented programming is used in the Python files, the compliance to AppInspect defined for the question regarding coding standards, the number of log files and the metric about the number of data points and output data fields used in a SPL query was not as expected. For example, the number of data points and output data fields are used to give an indication about the complexity of SQL queries. And because the SPL language is derived from the SQL language, we expected that these metrics were also useful in measuring the complexity of SPL queries. Furthermore, AppInspect is developed by Splunk to check whether a Splunk app complies to a certain standard such that it is allowed to publish it on Splunkbase so we expected that this metric would also be included. Lastly, the number of log files was also expected to be included in the model as this was considered as very useful by looking at my own experience. Because, it is useful to separate the logging in different groups which makes it easier to search through the log messages.

12.5 Misunderstanding during the validation

As mentioned in Section 7.2, some of the participants have not understood the main question of the block about the volume of the source code which has probably let to answering the statements in this block with a low score. Furthermore, we have also mentioned in this section that the metrics containing the word "token" received a lot of "neutral" answers which was probably caused by the problem around misunderstanding the word "token". By knowing this, it would be useful to redo the validation of these metrics as the results of our survey regarding these metrics are not reliable.

12.6 Real-world validation of the ratings

At the end of this master thesis, we have validated the quality model by conducting an empirical investigation at SMT. During this validation, we have presented our final quality model and explained how we are able to give a new Splunk app a certain maintainability score based on the benchmark rating systems. However, this does not validate whether a low or high maintainability score is indeed less or more maintainable in the real-world. In order to validate whether the quality model proposed in this thesis is indeed able to rate a Splunk app based on the maintainability, we could use the research conducted by Bijlsma et al. [Bij10]. In this research, he found a positive correlation between the maintainability and resolution times for enhancements and a stronger correlation for defects.

12.7 Validation of the model by using the tool

As explained in the aforementioned subsection, we have validated the final quality model by conducting an empirical investigation at SMT in which we have presenting the final quality model. However, we have not validated whether the maintainability score provided by our tool matches the score given by Splunk developers. To address this, we could compare the ranking of several Splunk

apps based on the maintainability scores provided by our tool to the ranking based on the opinions of the Splunk developers. However, we were not able to conduct this type of validation because we have not implemented the scripts and benchmark rating systems for all metrics in our quality model. Therefore, it is not possible to rank several Splunk apps based on the maintainability score provided by our tool because this is not implemented yet.

12.8 Generalization

The results obtained during this master thesis are specifically explained for and based on the Splunk platform. However, as mentioned in the introduction, Splunk is one of the larges big data monitoring platforms available on the market. Within the big data monitoring domain, there are many more alternative platforms such as Elastic [ela], Dynatrace [dyn] and Datadog [Dat16]. The quality model provided in this thesis could also be used for other platforms such as Elastic because Elastic has also dashboards with panels, drilldowns, searches and dashboard filters. Furthermore, it also has alert actions, fields and reports which are called knowledge objects within Splunk. However, not all of the defined quality metrics are directly usable for other platforms and should therefore be translated or replaced by another metric. This because the terminology used in the quality model is based on Splunk and should sometimes be translated to the terminology used within, for example, Elastic. Furthermore, the implementation of the tool and the benchmark can't be used for the other platforms as the implementation of the tool is based on the Splunk language and the benchmark is done by using a dataset containing the source code of Splunk apps. However, the method used to define the metric and rating thresholds can be used for the other platforms as well.

13 Conclusions and future work

In this master thesis, we have created a quality model and presented a design of how the tool should look like in the end. With this quality model and tool, we want to provide Splunk developers with instruments that can be used to continuously measure and monitor the maintainability of Splunk apps without the need of much time. In this chapter, we will give a short summary of the process of this thesis and provide answers to the defined research questions in Section 1.4.

13.1 Summary

First of all, we have created an initial quality model by doing literature research and using my own experience and thoughts regarding Splunk. During this step, we have investigated which components of a Splunk app are subjected to maintainability and how these components, that are subjected to maintainability, can be measured. In order to define quality metrics that are able to give an indication of the maintainability of Splunk apps, we have used the so-called goal-question-metric approach. A lot of quality metrics for measuring maintainability exists in the literature which makes it difficult to define the best set of quality metrics without much context. However, by using the goal-question-metric approach, which uses a top-down instead of bottom-up approach, we were able to mitigate this problem because goals and questions are first defined before the metrics are chosen. After defining the goals and question of our initial quality model, we searched for existing quality models by conducting literature research. We found a lot of quality models for programming languages such as Python and Java, however, none of them were especially created for Splunk. But because this master thesis is about the maintainability of Splunk apps that consist of the Splunk and Python language, we first defined the quality metrics for the Python code divided into groups based on which question they could answer by using these already existing quality models [MRW77][BBL76][iso]. After that, we have tried to translate these quality metrics such that it can be applied on the Splunk language. Furthermore, we have used our own experience and thoughts to define additional quality metrics, especially for Splunk, when needed.

Secondly, we have validated the initial quality model by conducting both two focus groups and a survey. The focus groups were used to validate the goals and questions defined in our initial quality model and the survey was used to validate the metrics defined in our initial quality model. The results of both the focus groups ans survey are implemented in our initial quality model which resulted in our pre-final quality model. The focus groups have let to the adjustment of the formulation of some questions and the addition of several new quality metrics for the Splunk language, one new question and a manual checklist that measures non-automatable metrics such as the existence of an architectural overview of the possible data flows, the existence of automated tests or sample data which can be found in Table 18. The survey has let to the exclusion of some quality metrics from our pre-final quality model by using a selection criteria that is based on the answers on the Likert-scale statements which we think have let to a good selection of metrics. Furthermore, participants of the survey indeed validated the need for such a tool and model that minimizes the time needed to map the maintainability of a Splunk app by mentioning the following: Splunk apps are written very fast to get something to the market immediately and then abandoned without any documentation from both Splunk and third party developers.

After that, we have created the Python scripts for automatically measuring the metrics defined in our quality model. However, because of time-constraints and the size of this master thesis, we have only implemented the scripts to measure the quality metrics defined for the volume question in our pre-final quality model. After implementing these scripts, we have created a benchmark dataset of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2). We were not able to use the Github API to automatically return and download suitable Splunk apps as it doesn't contain the source code of many Splunk apps. Therefore, we had to manually search and download suitable Splunk apps from Splunkbase which is the marketplace of Splunk. This dataset consists of a small fraction of the 700 available Splunk apps located on Splunkbase. The reason for this is that we have to check each Splunk app manually on the presence of `.conf` and `.xml` files. Furthermore, we have to review each app and remove third-party code manually which is a time consuming task. By using the Python scripts and benchmark dataset, we have created an empirically-based rating system for each quality metric defined for the volume question in our pre-final quality model. During the creation of the empirically-based rating system, we have deleted one quality metric from our pre-final quality model resulting in our final quality model. We have deleted this metric because the metric value is 0 for more than 70% of the total number of tokens in the dashboard files of our benchmark dataset. For the quality metrics having only one value per Splunk app such as the number of tokens in a Splunk app, it is not useful to create a benchmark rating system based on the method explained in Section 2.4 because of the distribution of these metrics explained in Section 10.2.

Last but not least, we have validated our final quality model and prototype of our tool for automatically measuring and monitoring the maintainability score of a Splunk app by conducting an empirical investigation at SMT. For this validation step, we have chosen to conduct three interviews with employees of SMT. In general, they mentioned that the model/tool is useful in practice but also shared some interesting thoughts which have been discussed in Section 11. For example, they indicated that the model/tool should be validated in practice because it is possible that a certain Splunk app is ranked with a very high score but isn't easy to maintain in practice. By implementing these suggestions, this model/tool is assessed as useful and ready to be used in practice.

13.2 Answers to the research questions

In this thesis we investigated the following questions:

1. *Which components of a Splunk app are subjected to maintainability?*
   In this master thesis, we have focused on Splunk apps that consist of the Splunk language in combination with the Python language. With this information, we are able to say that the following components of a Splunk app are subjected to maintainability:

   - SPL queries
   - Configuration files
   - Dashboards
   - Knowledge objects
   - Custom Python files

- Documentation

A Splunk app can also consist of other programming languages such as HTML, Javascript and more. However, these kind of Splunk apps are beyond the scope of this master thesis.

2. *How can we define good maintainable code for each component defined in the first question?*
In order to define good maintainable code for each component, we had to understand the term "Maintainability". The definition of maintainability used during this master thesis is the definition provided by the IEEE Standard Glossary of Software Engineering Terminology [Rad90] formulated as follows:

> *The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment* [Rad90].

This is a well-established and widely used [DJ03][HF02][Lan02] definition in the software development domain. After getting an understanding of the term "maintainability", we started to search for existing quality models by conducting literature research. Using this definition and existing quality models, we define good maintainable code as code with a high degree of modifiability, understandability and testability.

3. *Which quality metrics are suitable for measuring the aspects defined in the second question?*
In order to define quality metrics that are suitable for measuring the modifiability, understandability and testability, we first had to define questions that should indicate how the defined goals can be assessed and thereby indicate whether or not a specific goal is met according to the goal-question-metric approach. After defining the questions, we first defined the quality metrics for the Python code divided into groups based on which question they could answer by using already existing quality models [MRW77][BBL76][iso]. After that, we have tried to translate these quality metrics such that it can be applied on the Splunk language. Furthermore, we have used our own experience and thoughts to define additional quality metrics, especially for Splunk, when needed. This initial quality model was then validated by conducting both two focus groups and a survey which results are implemented in the initial quality model resulting in a pre-final quality model. The final quality model can be found in Appendix B.

4. *How can we define an empirically-based rating system for each quality metric from the third question?*
We implemented the Python scripts to automatically measure the quality metrics defined for the volume question in our pre-final quality model. After that, we have created a benchmark dataset of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2) for which we have measured the metric values using the implemented Python scripts. These metric values are first used to verify whether the metrics give useful information and are therefore a good or bad candidate for measuring the maintainability of a Splunk app. We assessed the usability of the metrics by looking at whether a metric is only present in a small number of Splunk apps and how the metric values are distributed. This assessment has let to the deletion of one quality metric from our pre-final quality model resulting in our final quality model. We have deleted this metric because the metric value is 0 for more than 70% of the total number of tokens in the dashboard files of our benchmark dataset.

After that, we have defined the metric thresholds which have been used to create a rating system per metric. The explanation of the methods used for defining the metric thresholds and rating systems can be found in Chapter 2.4. The implementation of these methods on the quality metrics defined for the volume question in our final quality model can be found in Chapter 10.2.

5. *How can these quality metrics from the third question be combined into a single quality model?*
We have only implemented the Python scripts, metric thresholds and rating systems for the quality metrics defined for the volume question in our quality model. Therefore, we only have a suggestion of how these quality metrics can be combined into a single quality model which is taking the average ratings of all metrics in our quality model as the maintainability score. However, we were not able to validate whether the average ratings gives a good indication of the maintainability score. For future work, this can be validated by comparing the ranking of Splunk apps done by Splunk experts to the ranking done by our tool.

6. *Is the quality model experienced as useful in practice by Splunk related companies?*
In general, the participants of the interviews indicated that the model/tool is useful in practice. However, they have shared interesting thoughts about the model and tool:

   (a) The model/tool should be validated in practice because it is possible that a certain Splunk app is ranked with a very high score but isn't easy to maintain in practice.

   (b) It should be possible to adjust the metric thresholds for certain metrics.

   (c) The maintainability score should be calculated by using the median of the ratings instead of the mean because the median is less sensitive to outliers.

   (d) A new functionality should be added to the model/tool such that already existing Splunk apps on Splunkbase can be rated by using this tool that can indicate which Splunk app is probably best to install.

   By implementing these suggestions, this model/tool is assessed as useful and ready to be used in practice.

7. *What is the effect of the newly created model on the strategic goals of a company?*
The quality model and created design are suitable for providing developers continuous insights into the maintainability of their Splunk app and thereby improving their development process and making better informed decisions based on valuable data which has also been validated by one of the interviewees of section 11. As mentioned before, organizations value systems and products that can be better adapted and improved to serve new business needs or to compensate for changes in the underlying systems which can be achieved by ensuring higher maintainability. By using the quality model and created design of the tool, the developers can use the provided information to improve the maintainability of their Splunk app and thereby develop more adaptable/changeable Splunk apps. Furthermore, the development of high maintainable code would reduce the maintainability costs because less time have to be spent on the maintainability activities. Lastly, during the validation process, one of the participants mentioned that the quality model and tool makes it possible to give clients better advice whether or not to install a specific app based on the maintainability score which

will, eventually, lead to time and costs savings. Furthermore, one of the participants also mentioned that at SMT they are responsible for the maintenance of Splunk apps that are installed at their clients and to solve issues that occur. Therefore, it is important to install apps at their clients that are easy to maintain which, eventually, saves them a lot of time and effort. However, this is based on the validation by conducting two focus groups, a survey and three final interviews in which I presented the quality model and created design. Therefore, it is still needed to create the tool and to validate the model by using the tool and to perform a real-world validation of the maintainability ratings which is also discussed in Section 12.

13.3 Future work

13.3.1 Future development

In this master thesis, we have created a quality model for measuring the maintainability score of Splunk apps. This quality model consists of three goals, 10 questions and multiple quality metrics. For each quality metric defined in our quality model, we should implement a script for automatically calculating the metric values. Furthermore, we should also create a benchmark rating system for each quality metric defined in our quality model. However, because of time-constraints and the size of this master thesis, we have decided to write the scripts and create the benchmark rating systems only for the quality metrics defined for the volume question in our quality model. For future development, it would be interesting to write the scripts and create the benchmark rating systems for the other metrics defined in our quality model. By doing this, we have translated the entire quality model into a tool that is able to automatically measure all metric values in our quality model. This makes it possible to give a Splunk app a rating for each metric based on the metric value and thereby give an overall maintainability score to a Splunk app.

Furthermore, the scripts that automatically measure the metrics that are related to Splunk dashboards can only be applied to dashboard files that uses the standard XML layout of Splunk. However, we have seen in our benchmark dataset that, in some cases, the dashboards are implemented by using Javascript instead. It would therefore be interesting to investigate how the metrics related to Splunk dashboards can be implemented for both options.

13.3.2 Test on larger benchmark dataset

In this master thesis, we have used a benchmark dataset containing the source code of 50 Splunk apps. However, during the process of defining the metric and rating thresholds, we encountered a problem which caused that for some metrics it was not able to fit a 5−30−30−30−5% distribution for the rating systems. Therefore, we have adjusted the percentages for the thresholds from 70/80/90% to 40/60/80%. This problem would probably not occur when using a larger benchmark dataset because the larger the benchmark dataset, the lower the contribution of one Splunk app on the overall score. We have not verified this thought because of time-constraints and the size of this master thesis but would, however, be interesting for future research.

13.3.3 Maintainability score by using weights

For defining the overall maintainability score, we have used the average of the ratings of all metrics in our quality model. However, it is likely that for some metrics it is more important to have a higher score than for other metrics in our quality model. Therefore, it would be interesting to add weights for each metric in our quality model based on the impact it has on the maintainability of a Splunk app.

13.3.4 Statistical analysis

In this master thesis, we have created a quality model containing metrics that can be measured in order to provide a Splunk app with a maintainability score. However, because of time-constraints and the size of this master thesis, we have not done a statistical analysis on the benchmark dataset in order to verify the usefulness of each metric. By conducting a statistical analysis, we could, for example, detect metrics that are correlated and should therefore not be included both in the quality model according to the MECE principle. In order to see which metrics are correlated, we could use the Spearman's correlation as it can be used to measure the correlation between two variables measured on at least an ordinal scale and it is already used in a similar type of research [vdB16]. This assessment could therefore lead to the deletion of some metrics which results in a different set of metrics compared to our final quality model.

References

[ACBV20] Luca Ardito, Riccardo Coppola, Luca Barbato, and Diego Verga. A tool-based perspective on software code maintainability metrics: a systematic literature review. *Scientific Programming*, 2020, 2020.

[ACV11] Tiago L Alves, José Pedro Correia, and Joost Visser. Benchmark-based aggregation of metrics to ratings. In *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 20–29. IEEE, 2011.

[AMM19] Gloria Arcos-Medina and David Mauricio. Aspects of software quality applied to the process of agile software development: a systematic literature review. *International Journal of System Assurance Engineering and Management*, 10(5):867–897, 2019.

[app] Validate quality of apps or add-ons with splunk appinspect for splunk cloud or splunk enterprise.

[ASC02] Krishan K Aggarwal, Yogesh Singh, and Jitender Kumar Chhabra. An integrated measure of software maintainability. In *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No. 02CH37318)*, pages 235–241. IEEE, 2002.

[Atk99] Roger Atkinson. Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International journal of project management*, 17(6):337–342, 1999.

[AYV10] Tiago L Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE, 2010.

[Bar00] J Jackson Barnette. Effects of stem and likert response option reversals on survey internal consistency: If you feel the need, there is a better alternative to using those negatively worded stems. *Educational and Psychological Measurement*, 60(3):361–370, 2000.

[BB12] Harry N Boone and Deborah A Boone. Analyzing likert data. *Journal of extension*, 50(2):1–5, 2012.

[BBL76] Barry W Boehm, John R Brown, and Mlity Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605, 1976.

[BCSV12] Robert Baggen, José Pedro Correia, Katrin Schill, and Joost Visser. Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, 20(2):287–307, 2012.

[Bij10] Dennis Bijlsma. *Indicators of issue handling efficiency and their relation to software maintainability*. PhD thesis, Master's thesis, University of Amsterdam, 2010.

[BJ95] Frederick P Brooks Jr. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[BOB+20] AMOS O Bajeh, ONILEDE-JACOBS Oluwatosin, SHUIB Basri, ABIMBOLA G Akintola, and ABDULLATEEF O Balogun. Object-oriented measures as testability indicators: an empirical study. *J. Eng. Sci. Technol*, 15:1092–1108, 2020.

[BW09] Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2009.

[CALO94a] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[CALO94b] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[Car14] David Carasso. *Exploring Splunk: search processing language (SPL) primer and cookbook*. CITO Research, 2014.

[CH17] Mingda Chen and Yao He. Exploration on automated software requirement document readability approaches, 2017.

[con] List of configuration files. Available at https://docs.splunk.com/Documentation/Splunk/8.1.3/Admin/Listofconfigurationfiles.

[CR94] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, pages 528–532, 1994.

[Dat16] Datadog. Datadog, Jul 2016. Available at https://www.datadoghq.com/.

[Diw16] IG Diwan. Predicting maintainability of object-oriented system using fuzzy logic. *International Journal of Scientific and Research Publications*, 6(3), 2016.

[DJ03] M. Dagpinar and J.H. Jahnke. Predicting maintainability with object-oriented metrics -an empirical comparison. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 155–164, 2003.

[dyn] The leader in cloud monitoring. Available at https://www.dynatrace.com/.

[ela] Free and open search: The creators of elasticsearch, elk kibana. Available at https://www.elastic.co/.

[Eme84] Thomas J Emerson. A discriminant metric for module cohesion. In *Proceedings of the 7th international conference on Software engineering*, pages 294–303, 1984.

[FN99] Norman E Fenton and Martin Neil. Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3):149–157, 1999.

[FN00] Norman E Fenton and Martin Neil. Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 357–370, 2000.

[Gil76] T. Gilb. *Software Metrics.* Chartwell-Bratt, 1976.

[GWMW08] Abdul Azim Abdul Ghani, KT Wei, Geoffrey M Muketha, and W Pei Wen. Complexity metrics for measuring the understandability and maintainability of business process models using goal-question-metric (gqm). 2008.

[Hal77] Maurice H Halstead. Elements of software science. 1977.

[HF02] A. Hajnal and I. Forgacs. A precise demand-driven definition-use chaining algorithm. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, pages 77–86, 2002.

[HKV07] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *6th international conference on the quality of information and communications technology (QUATIC 2007)*, pages 30–39. IEEE, 2007.

[Hol21] Arne Holst. Total data volume worldwide 2010-2024, Feb 2021.

[Inc] Splunk Inc. Dashboards and visualizations.

[iso] Iso/iec 25010:2011. (2011). systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. retrieved december 15, 2020, from https://www.iso.org/obp/ui/iso:std:iso-iec:25010:ed-1:v1:en.

[joo] Joost visser. Available at https://www.universiteitleiden.nl/en/staffmembers/joost-visser.

[Jva20] Jvanoel. Data-driven solution provider, Dec 2020. Available at https://www.smtware.com/.

[Kan03] Stephen H Kan. *Metrics and models in software quality engineering.* Addison-Wesley Professional, 2003.

[KBL08] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. The focus group method as an empirical tool in software engineering. In *Guide to advanced empirical software engineering*, pages 93–116. Springer, 2008.

[kno] What is splunk knowledge? Available at https://docs.splunk.com/Documentation/Splunk/8.1.3/Knowledge/WhatisSplunkknowledge.

[Lan02] Rikard Land. Measurements of software maintainability. In *Proceedings of the 4th ARTES Graduate Student Conference*, pages 1–7, 2002.

[Log] Logilab. code analysis for python: www.pylint.org.

[Lue17] Stephen Luedtke, Sep 2017. Available at https://conf.splunk.com/files/2017/slides/power-of-spl.pdf.

[Maz16] Sourav Mazumder. Big data tools and platforms. *Big Data Concepts, Theories, and Applications*, page 29–128, 2016.

[MBWW20] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. An empirical validation of cognitive complexity as a measure of source code understandability. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2020.

[McC76] A McCabe. Complexity measure. *IEEE Transaction on Software Engineering*, pages 308–320, 1976.

[mcg] Betere ondersteuning, tevreden klanten. Available at https://www.topdesk.com/nl/.

[MD06] Ernest Mnkandla and Barry Dwolatzky. Defining agile software quality assurance. In *2006 International Conference on Software Engineering Advances (ICSEA'06)*, pages 36–36. IEEE, 2006.

[Min09] Barbara Minto. *The pyramid principle: logic in writing and thinking*. Pearson Education, 2009.

[MJ06] Peter Meso and Radhika Jain. Agile software development: adaptive systems principles and best practices. *Information systems management*, 23(3):19–30, 2006.

[MKL17] Philip Mayer, Michael Kirsch, and Minh Anh Le. On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. *Journal of Software Engineering Research and Development*, 5(1):1–33, 2017.

[Mor96] David L Morgan. Focus groups. *Annual review of sociology*, 22(1):129–152, 1996.

[MR04] Radu Marinescu and Daniel Ratiu. Quantifying the quality of object-oriented design: The factor-strategy model. In *11th Working Conference on Reverse Engineering*, pages 192–201. IEEE, 2004.

[MRW77] Jim A McCall, Paul K Richards, and Gene F Walters. Factors in software quality. volume-iii. preliminary handbook on software quality for an acquisiton manager. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA, 1977.

[OLSB09] Philipp Offermann, Olga Levina, Marten Schönherr, and Udo Bub. Outline of a design science research process. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, pages 1–11, 2009.

[OPR03] Maryoly Ortega, María Pérez, and Teresita Rojas. Construction of a systemic quality model for evaluating a software product. *Software Quality Journal*, 11(3):219–242, 2003.

[PMDL99] J-F Patenaude, Ettore Merlo, Michel Dagenais, and Bruno Laguë. Extending software quality assessment techniques to java systems. In *Proceedings Seventh International Workshop on Program Comprehension*, pages 49–56. IEEE, 1999.

[pyt] splunk-addon documentation¶. Available at https://pytest-splunk-addon.readthedocs.io/en/latest/.

[PZS83] G. Parikh, N. Zvegintzov, and IEEE Computer Society. *Tutorial on Software Maintenance*. IEEE computer Society tutorials. IEEE Computer Society Press, 1983.

[Rad90] Jane Radatz. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[RB90] HD Rombach and VR Basili. Benefits of goal oriented measurement. In *Proc 7th CSR Annual Conference on Software Reliability and Metrics*, 1990.

[rev14] Big data: The management revolution, Oct 2014.

[Rom02] H Dieter Rombach. Software quality versus time-to-market: How to resolve these conflicts? In *European Conference on Software Quality*, pages 1–1. Springer, 2002.

[SBV+17] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. Automatically assessing code understandability: How far are we? In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 417–427. IEEE, 2017.

[sea] List of search commands. Available at https://docs.splunk.com/Documentation/SplunkLight/7.3.6/References/Listofsearchcommands.

[SLVPO16] Simone Scalabrino, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016.

[Spi06] Diomidis Spinellis. *Code quality: the open source perspective*. Adobe Press, 2006.

[spla] The data-to-everything platform built for the cloud. Available at https://www.splunk.com.

[splb] Developer guide for splunk cloud and splunk enterprise. Available at https://dev.splunk.com/enterprise/docs/welcome.

[splc] Spl in splunk. Available at https://docs.splunk.com/Splexicon:SPL.

[SR18] Made Agus Putra Subali and Siti Rochimah. A new model for measuring the complexity of sql commands. In *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–5. IEEE, 2018.

[SRE18] Reina Setiawan, Zulfany Erlisa Rasjid, and Andy Effendi. Design metric indicator to improve quality software development (study case: Student desk portal). *Procedia Computer Science*, 135:616–623, 2018.

[vdB16] ECH van der Bent. Defining and measuring puppet code quality. Master's thesis, 2016.

[VDBvdL07] Huib J Van Den Brink and Rob van der Leek. Quality metrics for sql queries embedded in host languages. In *Eleventh European Conference on Software Maintenance and Reengineering (CSMR 2007). Software Improvement Group*. Citeseer, 2007.

[WHW91] Richard Widdows, Tia A Hensler, and Marlaya H Wyncott. The focus group interview: A method for assessing users' evaluation of library service. 1991.

[YC88] Stephen S Yau and Pao-Sheng Chang. A metric of modifiability for software maintenance. In *1988 Conference on Software Maintenance*, pages 374–381. IEEE Computer Society, 1988.

A Focus group schedule

The duration of the focus group is around 2 hours and contains the following schedule:

Time	Subject
10 min	Structure and goals of the focus group
5 min	Introduction of the participants
10 min	Introduction to the topic
15 min	Group discussion about the question: Why should we want to improve the maintainability of Splunk apps?
10 min	Goals of the quality model
10 min	Group discussion about the question: What do you think of these goals?
15 min	Group discussion about the question: Which questions should we answer to make those goals measurable?
10 min	Questions of the quality model
10 min	Group discussion about the question: What do you think of these questions?
15 min	General questions
10 min	Closing
2 hours	Total

Table 25: The schedule for the two focus groups with a duration of approximately 2 hour.

B Final quality model

This is the final quality model after the adjustments of the initial quality model by using the results of both two focus groups and a survey.

B.1 Final quality model: Goals

Goal	Purpose	Issue	Object	Viewpoint
Improve the speed to understand the software from the development teams' viewpoint	Improve	The speed to understand	The software	Development team
Improve the easiness to test the software from the development teams' and consultants' viewpoint	Improve	The easiness to test	The software	Development team and consultants
Increase the effectiveness and efficiency to modify the software from the development teams' viewpoint	Increase	The effectiveness and efficiency to modify	The software	Development team

Table 26: The defined goals using the structure of the goal-question-metric approach.

B.2 Final quality model: Questions

Question	Motivation
To what extent do you comply with the **coding guidelines** that guide you how to structure the code of your Splunk app?	By using clear requirements or coding guidelines, developers not familiar with the project can better understand the code.
What is the **volume** of the code? [HKV07]	According to the SIG quality model, the total size of a system should feature heavily in any measure of maintainability. A larger system requires, in general, a larger effort to maintain. Furthermore, the larger a function or file is, the more actions/functionalities it contains and thus the more difficult it is to understand that function or file.
What is the **readability** of the code? [BW09] [SBV+17]	The readability of a program is related to its maintainability, and is thus a key factor in overall software quality. Furthermore, a low readability will also negatively affect the understandability of the code.
Does descriptive **documentation** exist for the software? [SBV+17]	A good and clear documentation will help maintainers, who are not familiar with the software, to understand the software and code by explaining the functionalities and different parts in the code.

Table 27: The defined questions for the first goal (understandability) including the motivation of defining this question is specified in this table.

Question	Motivation
Is the **logging** instrumentation inplace?	One of Splunk's most important applications is the processing of log data and use it to create better insights into, for example, the performance of code. Furthermore, it will lower the MTTR because the logs give a good indication of the location of the problem and the problem itself.
What is the **volume** of the code? [HKV07]	According to the SIG quality model, the total size of a system also affects the testability of a system. Furthermore, the larger a function or file is, the more actions/functionalities it contains and thus the more difficult it is to test all possibilities of that function or file.
What is the **complexity** of the source code? [HKV07]	According to the SIG quality model, the complexity of the source code units influences the system's changeability and its testability.
Are there **automated tests** available?	According to the focus group sessions, the existence of automated tests is important because this gives a sense of safety, does the code work as it should work?

Table 28: The defined questions for the first goal (testability) including the motivation of defining this question is specified in this table.

Question	Motivation
How well is the code **structured**? [BBL76]	Modifiability consists of the following sub-characteristics: 1. Structuredness 2. Augmentability A more structured Splunk app will increase the ease to modify that Splunk app.
What is the amount of **duplication** in the code? [HKV07]	According to the SIG quality model, the degree of source code duplication (also called code cloning) influences analysability and changeability. Furthermore, the more duplication in the code, the more time it takes to modify the code because the changes have to be made to all duplicate code fragments.
What is the **complexity** of the source code? [HKV07]	According to the SIG quality model, the complexity of the source code units influences the system's changeability and its testability.
What is the **volume** of the source code?	The larger a file or unit is, the more difficult it is to modify that unit or file because the changes have effect on multiple fragments of that function or file.
What is the degree of **genericity** of the source code?	The more hardcodes, the more difficult it is to modify the source code because it is difficult to know where you have to make some changes.

Table 29: The defined questions for the first goal (modifiability) including the motivation of defining this question is specified in this table.

B.3 Final quality model: Metrics

Question	Metric Python	Metric Splunk
To what extent do you comply with the **coding guidelines** that guide you how to structure the code of your Splunk app?	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
	# of descriptive variable names (use of nouns)	# of descriptive variable names
	# of descriptive function names (use of verbs)	# of descriptive panel names
What is the **volume** of the code?	# of tokens in a function	# of tokens in a SPL query
	# of tokens in a file	# of tokens in a file
	# of tokens in a Splunk app	# of tokens in a Splunk app
	# of files in a Splunk app	# of files in a Splunk app
	# of statements in a function	# of commands in a SPL query
	Tiobe fan-out score of a file	# of drilldowns
	# of methods in a file	# of views
	# of external dependencies in a function	# of stanzas in a file
		# of Splunk knowledge objects
		# of fields used in a SPL search
		# of data sources used in a SPL search
		# of dashboard tokens ($tokens$) on a Splunk dashboard
		# of panels on a Splunk dashboard
		# of technologies used in a Splunk app (Python, java, css, html, config etc.)
		# of key-value pairs in a `.conf` file
What is the **readability** of the code?	# of tokens in a file	# of tokens in a file
	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
	# of descriptive variable names (use of nouns)	# of descriptive variable names
	# of descriptive function names (use of verbs)	# of descriptive panel names
	# of nested statements in a function	# of nested searches in a SPL query
	# of identifiers in a file	# of commands in SPL query
	# of '.' characters in a function	
	Cohesion of a method	
Does descriptive **documentation** exist for the software?	Gunning Fog Index of the documentation	Gunning Fog Index of the documentation
	The existence of a documentation	The existence of a documentation
	Comments and Identifiers Consistency in a file	Comments and Identifiers Consistency in a file
	# of comment lines in code (CLOC)	# of comment lines in code (CLOC)

Table 30: The defined metrics per question for the goal about understandability are specified in this table.

Question	Metric Python	Metric Splunk
Is the **logging** instrumentation inplace?	# of logging statements per function	
	# of logging statements that contains variables of that function	
What is the **volume** of the code?	Same metrics as in first question	Same metrics as in first question
What is the **complexity** of the source code?	Same metrics as in third question	Same metrics as in third question
Are there **automated tests** available?	included in the manual checklist	

Table 31: The defined metrics per question for the goal about testability are specified in this table.

Question	Metric Python	Metric Splunk
How well is the code **structured**?	# of tokens in a function	# of tokens in a SPL query
	# of tokens in a file	# of "order of execution" errors
	# of tokens containing formatting errors with the PEP-8 coding standard	# of tokens containing formatting errors with our Splunk coding standard
		% of compliance with Splunk app structure (Using AppInspect)
What is the amount of **duplication** in the code?	Tiobe duplication score	# of duplication between SPL queries on a dashboard or config file
What is the **complexity** of the source code?	Cyclomatic complexity of a function	# of nested searches
	Tiobe fan-out score of a file	# of custom configuration files
	Cohesion of a method	# of input data fields in a SPL query
	Coupling of a file	# of custom Python files
		# of different SPL commands
What is the **volume** of the source code?	Same metrics as in first question	Same metrics as in first question
What is the degree of **genericity** of the source code?	# of hardcoded strings in a Python function	# of hardcodes in a SPL query
	# of hardcoded integers in a Python function	
	# of duplicated hardcoded variables in a Python file	

Table 32: The defined metrics per question for the goal about modifiability are specified in this table.

B.4 Final quality model: Manual checklist

Question	Answer (Y/N)
Is there a CICD pipeline for your Splunk app?	
Is there a version control system (e.g. Bitbucket, github) for your Splunk app?	
Is there a visualisation of the data flows for your Splunk app?	
Are there unit tests available for your Splunk app?	
Are there functional tests available for your Splunk app?	
Is there sample data available for your Splunk app?	
Is there a monitoring dashboard for your Splunk app?	
Is there a documentation available for your Splunk app?	
How many add-ons are needed for using your Splunk app?	

Table 33: A manual checklist that measures non-automatable metrics such as the existence of an architectural overview of the possible data flows.

C Cumulative risk profiles

An example of how the cumulative risk profiles looks like by using the risk profiles of the metric about the number of tokens in a SPL query:

D Metric thresholds

This section contains the metric thresholds for the metrics defined for the volume question in our quality model, derived by using the benchmark dataset of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2).

D.1 Metric 1: the number of tokens in a SPL query

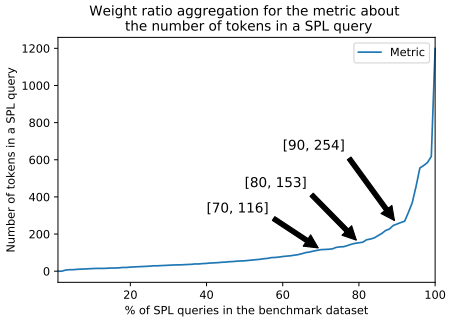


Figure 34: The metric distributions for the metric about the number of tokens in a SPL query.

Risk category	Number of tokens
Low	0 - 116
Moderate	117 - 153
High	154 - 254
Very high	255+

Table 34: The metric thresholds for the metric about the number of tokens in a SPL query.

D.2 Metric 2: the number of tokens in a file

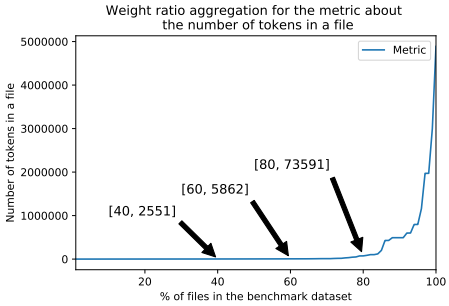


Figure 35: The metric distributions for the metric about the number of tokens in a file.

Risk category	Number of tokens
Low	0 - 2551
Moderate	2552 - 5862
High	5863 - 73,591
Very high	73,592+

Table 35: The metric thresholds for the metric about the number of tokens in a file.

D.3 Metric 3: the number of tokens in a Splunk app

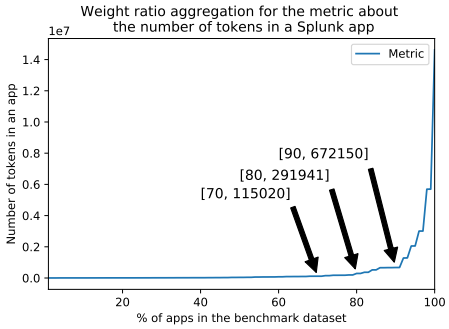


Figure 36: The metric distributions for the metric about the number of tokens in a Splunk app.

Risk category	Number of tokens
Low	0 - 115,020
Moderate	115,021 - 291,941
High	291,942 - 672,150
Very high	672,151+

Table 36: The metric thresholds for the metric about the number of tokens in a Splunk app.

D.4 Metric 4: the number of files in a Splunk app

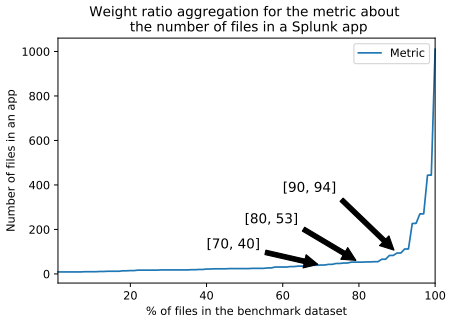


Figure 37: The metric distributions for the metric about the number of files in a Splunk app.

Risk category	Number of files
Low	0 - 40
Moderate	41 - 53
High	54 - 94
Very high	95+

Table 37: The metric thresholds for the metric about the number of files in a Splunk app.

D.5 Metric 5: the number of commands in a SPL query

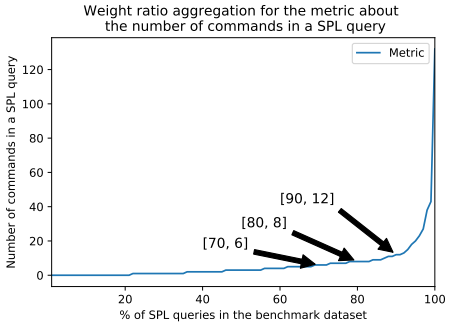


Figure 38: The metric distributions for the metric about the number of commands in a SPL query.

Risk category	Number of commands
Low	0 - 6
Moderate	7 - 8
High	9 - 12
Very high	13+

Table 38: The metric thresholds for the metric about the number of commands in a SPL query.

D.6 Metric 6: the number of dashboard tokens on a Splunk dashboard

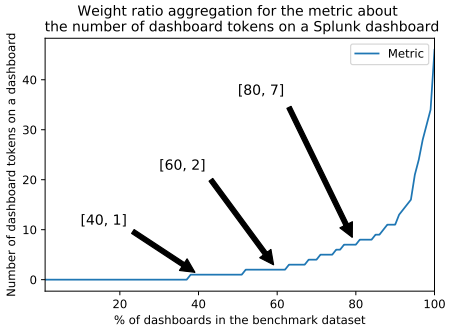


Figure 39: The metric distributions for the metric about the number of dashboard tokens on a Splunk dashboard.

Risk category	Number of dashboard tokens
Low	0 - 1
Moderate	2 - 2
High	3 - 7
Very high	8+

Table 39: The metric thresholds for the metric about the number of dashboard tokens on a Splunk dashboard.

D.7 Metric 7: the number of drilldowns on a Splunk dashboard

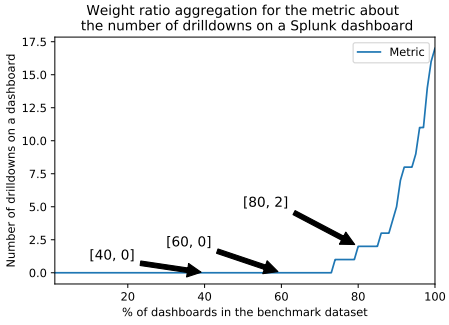


Figure 40: The metric distributions for the metric about the number of drilldowns on a Splunk dashboard.

Risk category	Number of drilldowns
Low	0 - 0
Moderate	0 - 0
High	1 - 2
Very high	3+

Table 40: The metric thresholds for the metric about the number of drilldowns on a Splunk dashboard.

D.8 Metric 8: the number of views in a Splunk app

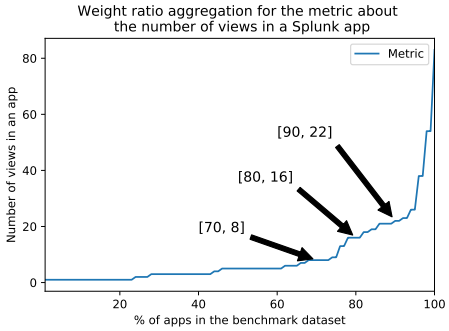


Figure 41: The metric distributions for the metric about the number of views in a Splunk app.

Risk category	Number of views
Low	0 - 8
Moderate	9 - 16
High	17 - 22
Very high	23+

Table 41: The metric thresholds for the metric about the number of views in a Splunk app.

D.9 Metric 9: the number of stanzas in a `.conf` file

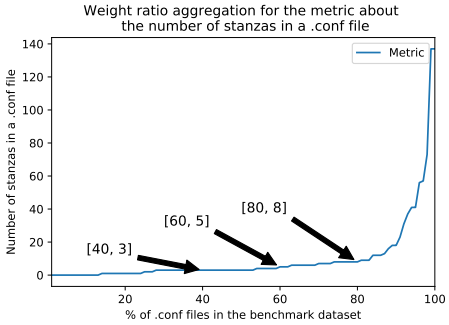


Figure 42: The metric distributions for the metric about the number of stanzas in a `.conf` file.

Risk category	Number of stanzas
Low	0 - 3
Moderate	4 - 5
High	6 - 8
Very high	9+

Table 42: The metric thresholds for the metric about the number of stanzas in a `.conf` file.

D.10 Metric 10: the number of knowledge objects in a Splunk app

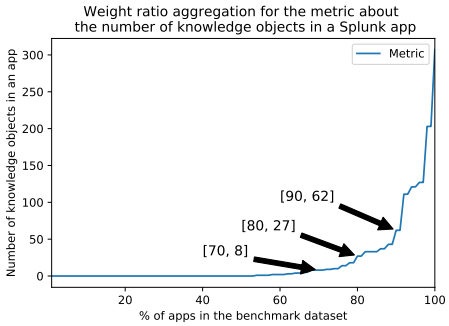


Figure 43: The metric distributions for the metric about the number of knowledge objects in a Splunk app.

Risk category	Number of knowledge objects
Low	0 - 8
Moderate	9 - 27
High	28 - 62
Very high	63+

Table 43: The metric thresholds for the metric about the number of knowledge objects in a Splunk app.

D.11 Metric 11: the number of panels on a Splunk dashboard

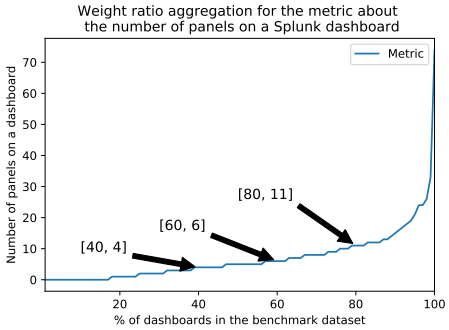


Figure 44: The metric distributions for the metric about the number of panels on a Splunk dashboard.

Risk category	Number of panels
Low	0 - 4
Moderate	5 - 6
High	7 - 11
Very high	12+

Table 44: The metric thresholds for the metric about the number of panels on a Splunk dashboard.

D.12 Metric 12: the number of technologies used in a Splunk app

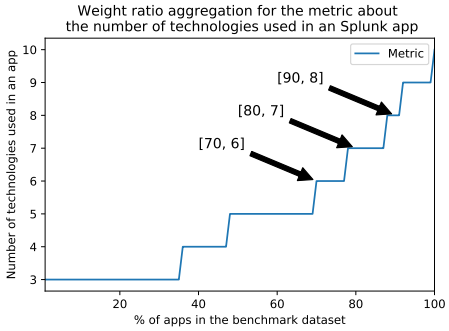


Figure 45: The metric distributions for the metric about the number of technologies used in a Splunk app.

Risk category	Number of technologies
Low	0 - 6
Moderate	7 - 7
High	8 - 8
Very high	9+

Table 45: The metric thresholds for the metric about the number of technologies used in a Splunk app.

D.13 Metric 13: the number of key-value pairs in a `.conf` file

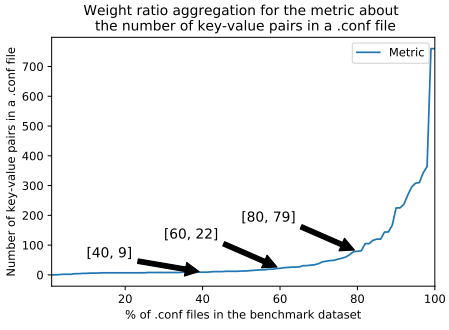


Figure 46: The metric distributions for the metric about the number of key-value pairs in a `.conf` file.

Risk category	Number of key-value pairs
Low	0 - 9
Moderate	10 - 22
High	23 - 79
Very high	80+

Table 46: The metric thresholds for the metric about the number of key-value pairs in a `.conf` file.

E Rating thresholds

This section contains the rating thresholds for the metrics defined for the volume question in our quality model, derived by using the benchmark dataset of 50 Splunk apps that I tried to select as random as possible (see also Section 12.2).

E.1 Metric 1: the number of tokens in a SPL query

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	13.2%	0%	0%
***	35.7%	24.0%	0%
**	70.2%	54.8%	32.0%
*	–	–	–

Table 47: The rating thresholds for the metric about the number of tokens in a SPL query.

E.2 Metric 2: the number of tokens in a file

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	52.5%	0%	0%
***	75.7%	45.4%	0%
**	98.6%	98.6%	94.5%
*	–	–	–

Table 48: The rating thresholds for the metric about the number of tokens in a file.

E.3 Metric 3: the number of tokens in a Splunk app

Rating	Number of tokens
*****	$0 - 5439$
****	$5439 - 16.990$
***	$16.990 - 97.228$
**	$97.228 - 3.008.490$
*	$> 3.008.490$

Table 49: The optimized rating thresholds used for the metrics with one metric value per Splunk app which is, in this table, the metric about the number of tokens in a Splunk app.

E.4 Metric 4: the number of files in a Splunk app

Rating	Number of files
*****	$0 - 10$
****	$10 - 19$
***	$19 - 37$
**	$37 - 270$
*	> 270

Table 50: The optimized rating thresholds used for the metrics with one metric value per Splunk app which is, in this table, the metric about the number of files in a Splunk app.

E.5 Metric 5: the number of commands in a SPL query

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	10.4%	0%	0%
***	44.0%	28.7%	0%
**	64.9%	55.4%	46.0%
*	–	–	–

Table 51: The rating thresholds for the metric about the number of commands in a SPL query.

E.6 Metric 6: the number of dashboard tokens on a Splunk dashboard

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	42.0%	6.5%	0%
***	81.2%	62.5%	20.3%
**	1%	1%	47.1%
*	–	–	–

Table 52: The rating thresholds for the metric about the number of dashboard tokens on a Splunk dashboard.

E.7 Metric 7: the number of drilldowns on a Splunk dashboard

This metric has been removed from our quality model because the metric values derived from the benchmark dataset were not as useful as we thought.

E.8 Metric 8: the number of views in a Splunk app

Rating	Number of views
*****	$0 - 1$
****	$1 - 4$
***	$4 - 7$
**	$7 - 38$
*	> 38

Table 53: The optimized rating thresholds used for the metrics with one metric value per Splunk app which is, in this table, the metric about the number of views in a Splunk app.

E.9 Metric 9: the number of stanzas in a `.conf` file

Rating	Moderate	High	Very high
*****	20.6%	0%	0%
****	65.9%	0%	0%
***	85.4%	52.0%	0%
**	99.2%	98.5%	97.3%
*	–	–	–

Table 54: The rating thresholds for the metric about the number of stanzas in a `.conf` file.

E.10 Metric 10: the number of knowledge objects in a Splunk app

Rating	Number of knowledge objects
*****	$0 - 0$
****	$0 - 0$
***	$0 - 5$
**	$5 - 127$
*	> 127

Table 55: The optimized rating thresholds used for the metrics with one metric value per Splunk app which is, in this table, the metric about the number of knowledge objects in a Splunk app.

E.11 Metric 11: the number of panels on a Splunk dashboard

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	42.3%	0%	0%
***	85.7%	55.3%	0%
**	96.0%	96.0%	77.5%
*	–	–	–

Table 56: The rating thresholds for the metric about the number of panels on a Splunk dashboard.

E.12 Metric 12: the number of technologies used in a Splunk app

Rating	Number of technologies
*****	$0 - 3$
****	$3 - 4$
***	$4 - 6$
**	$6 - 10$
*	> 10

Table 57: The optimized rating thresholds used for the metrics with one metric value per Splunk app which is, in this table, the metric about the number of technologies used in a Splunk app.

E.13 Metric 13: the number of key-value pairs in a `.conf` file

Rating	Moderate	High	Very high
*****	0%	0%	0%
****	68.1%	0%	0%
***	89.4%	53.0%	0%
**	98.4%	97.3%	93.6%
*	–	–	–

Table 58: The rating thresholds for the metric about the number of key-value pairs in a `.conf` file.

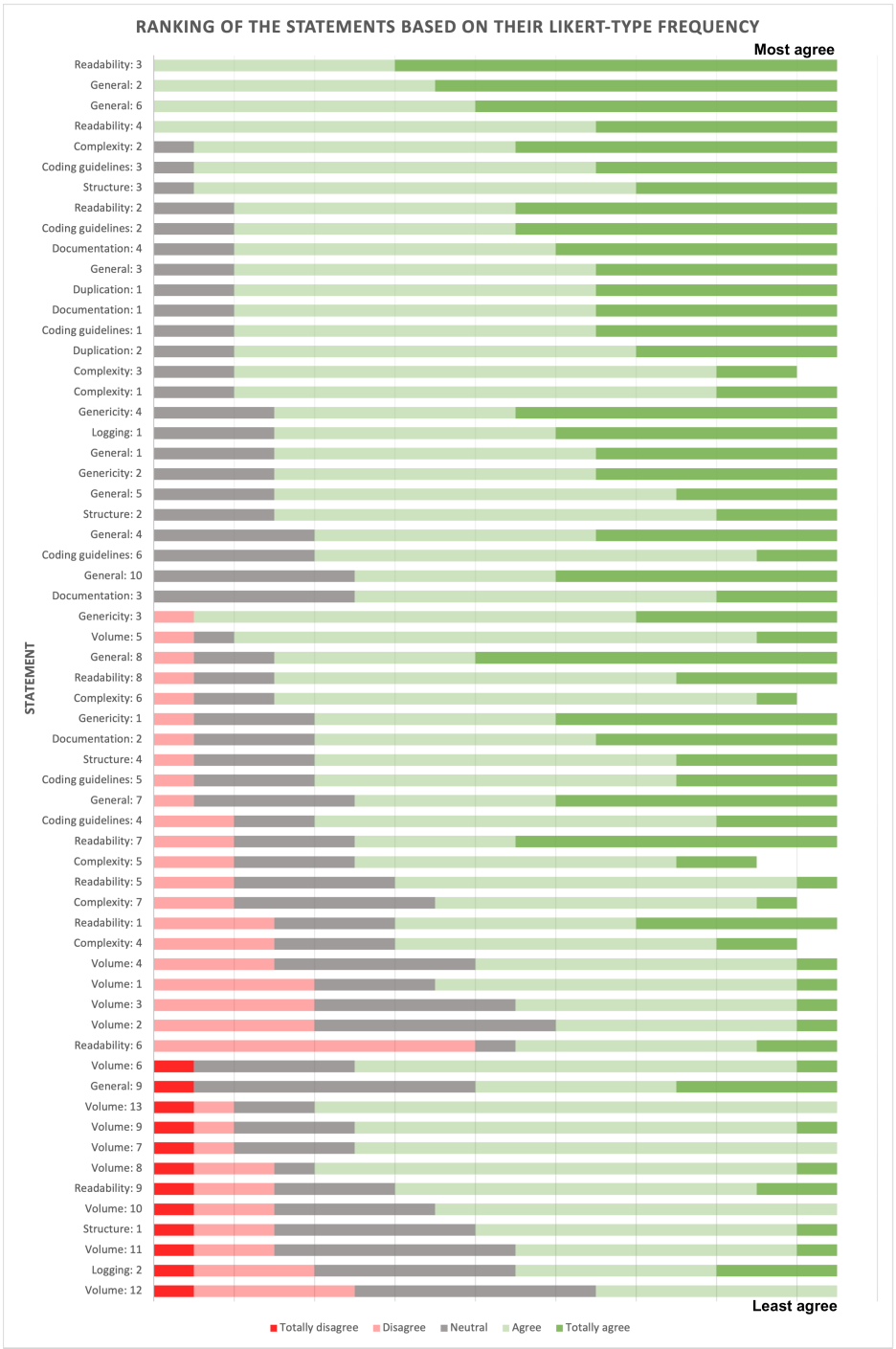


Figure 25: The ranking of the statements based on the frequencies of totally disagree, disagree, neutral, agree and totally agree. First, we have sorted the statements based on the frequency of totally agree, then agree, neutral, disagree and at last on the frequency of totally disagree. The statement number corresponds to the number in Table 17.

Moderate	High	Very high
1.0	0.8669250645994832	0.7344961240310077
0.8669250645994832	0.6654929577464789	0.6349136021267169
0.8253164556962025	0.6349136021267169	0.5277777777777778
0.7111727416798732	0.60249343832021	0.5157232704402516
0.7018779342723005	0.5761124121779859	0.380492515692902
0.6885245901639344	0.5537974683544303	0.31973058637083995
0.6646132785763176	0.5475435816164818	0.28823529411764703
0.6371391076115485	0.5157232704402516	0.2751137556877844
0.6349136021267169	0.4996577686516085	0.24727838258164853
0.572059413890004	0.46943942133815547	0.2106299212598425
0.5537974683544303	0.4572670207629165	0.1759069130732375
0.5393769688484424	0.42289504036908876	0.1697653545326324
0.5037878787878788	0.3566678333916696	0.14119259766963674
0.4756632064590542	0.3445467422096317	0.12613827993254637
0.4572670207629165	0.3344120819848975	0.10705244122965642
0.4441176470588235	0.3181115879828326	0.05852737570799245
0.4177545691906005	0.2860734037204625	0.04680851063829787
0.3930594900849858	0.2777279138351071	0.04474610356963298
0.3907296137339056	0.24727838258164853	0.01912739965095986
0.3790229516604693	0.23979763912310287	0.0
0.3745600804424334	0.1987662782727896	0.0
0.35674536256323774	0.17042553191489362	0.0
0.3344120819848975	0.16579634464751958	0.0
0.2925413130861992	0.11768407803650094	0.0
0.24727838258164853	0.07434944237918216	0.0
0.23214285714285715	0.05808027923211169	0.0
0.19989361702127662	0.04792540792540793	0.0
0.1987662782727896	0.04363753905829113	0.0
0.19076923076923077	0.0	0.0
0.15497168030207678	0.0	0.0
0.14312267657992567	0.0	0.0
0.13382178644542614	0.0	0.0
0.13179755671902268	0.0	0.0
0.09367493995196156	0.0	0.0
0.07720178372352285	0.0	0.0
0.03298611111111111	0.0	0.0
0.014417049728374426	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.0

Figure 33: An example of how the cumulative risk profiles looks like by using the risk profiles of the metric about the number of tokens in a SPL query.