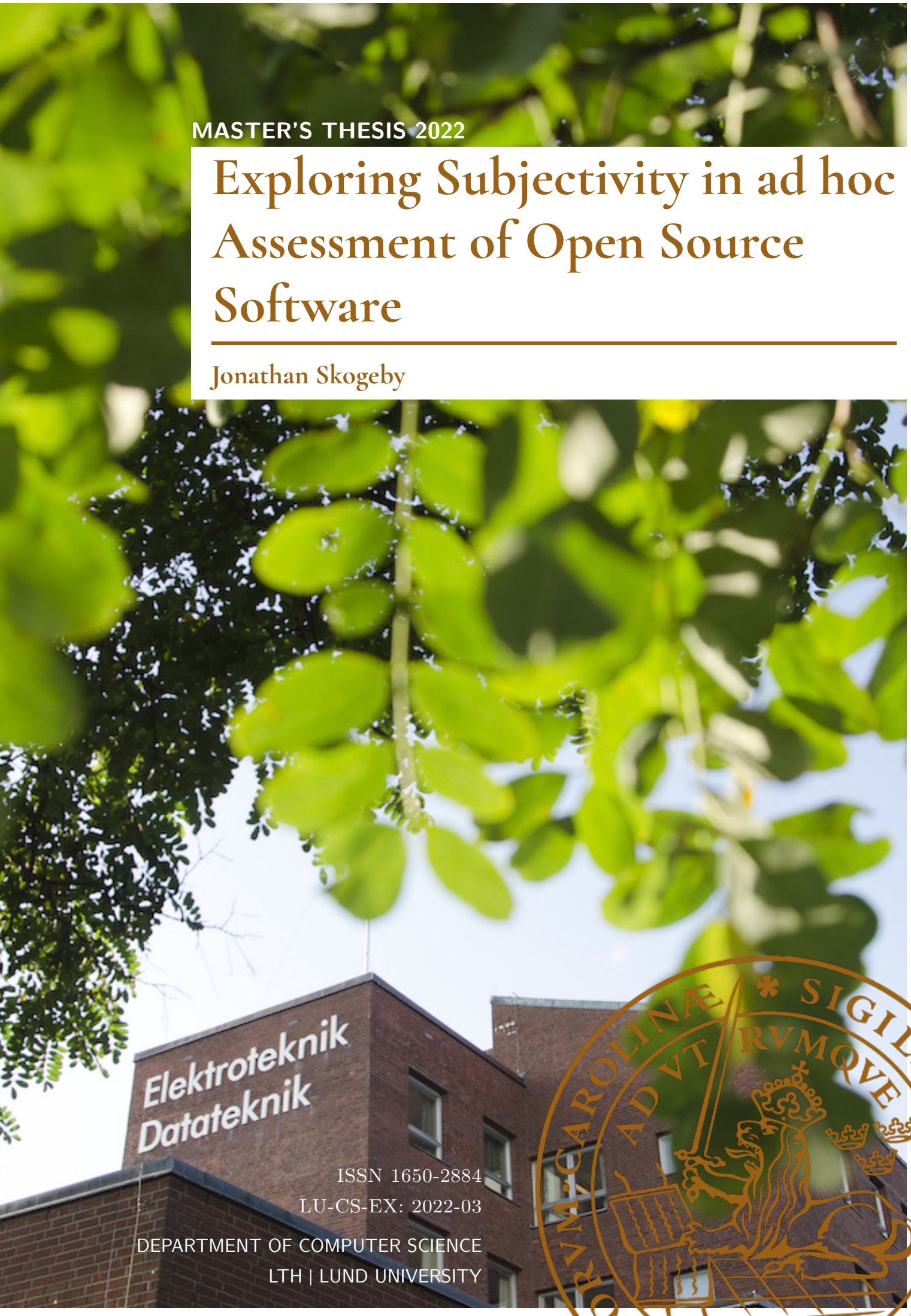


MASTER'S THESIS 2022

Exploring Subjectivity in ad hoc Assessment of Open Source Software

Jonathan Skogeby

ISSN 1650-2884
LU-CS-EX: 2022-03

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-03

Exploring Subjectivity in ad hoc Assessment of Open Source Software

Jonathan Skogeby

Exploring Subjectivity in ad hoc Assessment of Open Source Software

Jonathan Skogeby
jonathan.skogeby@gmail.com

February 17, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Martin Höst, martin.host@cs.lth.se
Emil Wåreus, emil.wareus@debricked.com

Examiner: Ulf Asklund, ulf.asklund@cs.lth.se

Abstract

Recent decades have seen a staggering increase in the presence of open source in modern software. Once approached with structured evaluation and selection processes in the form of quality assessment models that saw little adoption, contemporary methods have been observed to employ a more automated and data-driven philosophy. In alignment with this trend, this paper aims to explore the subjectivity involved in ad hoc open source assessment processes. Five developers were asked to rank twenty projects on the social coding platform GitHub in terms of two aspects: popularity and quality of contributors. In each aspect, the developers were subsequently instructed to list and order the most important metrics used in their evaluation. This data was used to examine how well their subjective opinion could be represented using a metric based linear model using data from the GitHub API. 22 metrics were defined and systematically tested in different combinations, with 10 000 randomized sets of weights tested for each combination. By applying Spearman's rank correlation coefficient, the combinations of metrics and their weights with the highest statistically significant correlation to each developer's ranking of each aspect were chosen for analysis. In total, ten sets of weighted metrics were generated (two sets for each of the five developers) and qualitatively compared against the developers' own narrative about which metrics were important in their evaluation. Although a high average Spearman correlation coefficient of 0.84 was achieved, the weighted metrics generated by the model showed very little resemblance to developers' reported metrics and order of importance. It was concluded that the implemented model was not successful in representing the developers' opinion, but discussion did not rule out the possibility of better results in potential future work.

Keywords: case study, open source software, GitHub, component assessment and selection, open source metrics, component based software development

Acknowledgements

Due to personal matters stemming from the COVID-19 pandemic, this thesis took considerably longer to finish than originally intended. I would first and foremost like to thank Emil Wåreus for his support and enthusiasm throughout this endeavor. I wish to show my gratitude to the amazing people of Debricked for making me feel as part of the team and making every day at the office a pleasure. Last but not least, a special thanks to Martin Höst for his patience and feedback on this paper.

Contents

Chapter 1

Introduction

The field of software engineering has long been pursuing ways to define quality in products and its source code in the ongoing information revolution. Once focused on addressing factors such as the six quality criteria of the ISO 9126 standard (functionality, reliability, usability, efficiency, maintainability, portability) [15], the rise of open source and an increase in component based software development has resulted in increased complexity of what defines quality in software.

Since the birth of the Open Source Definition, the prevalence of Open Source Software (OSS) has been rapidly increasing. In its infancy, OSS was approached with doubt and skepticism by many organizations due to concerns regarding security and competitor advantage [25]. Today, OSS is a cornerstone of modern software development. Black Duck, an organization dealing with logistics and legal matters surrounding OSS, revealed in a 2015 report that 66% of companies surveyed provide software for their customers that is built on OSS [10].

In addition, large platforms such as GitHub has facilitated the use of OSS, as well as the reuse of, and contribution to, its code. Even distinguished OSS projects predating GitHub such as the Linux kernel has opted to deploy mirror repositories on the platform to increase exposure [11]. With the current development paradigm characterized by reuse and component based software development (CBSD), OSS projects continue to grow in numbers and engagement. GitHub is as of writing home to the code of over 200 million publicly accessible software projects [7].

Despite the enormous presence of OSS in today's software, stakeholders are still trying to figure out how to optimize the selection process. This has been proven difficult for several reasons [26]. Many of these reasons have their root in the same cause, which is that metrics in OSS differ in importance depending on what is being evaluated and who is using it. Plainly, companies prioritize different key metrics and these metrics also vary depending on

the domain in which the OSS component resides. A start-up may focus on metrics resulting in decreased time to market, while a company in banking has strict requirements on security metrics. Likewise, an encryption library is likely to have different metrics of relevance than a machine learning library.

Thus, the nature of this selection process optimization has in recent times shifted focus, from finding the best alternative following rigorous evaluation in proposed quality models, to acknowledging the constraints and subjectivity in practitioners' unique situations by instead providing automated tools for analyzing relevant metrics. However, the subjectivity involved in the mainly experience oriented approach taken by software developers when assessing OSS is poorly covered in literature.

In an effort to begin filling this gap, this thesis explores the subjective nature of ad hoc OSS assessment. More specifically, we look into similarities and differences between the opinion of five surveyed developers and a simple metric based model, when ranking a set of open source projects. All chosen projects are from the social coding platform GitHub, and the participants of the survey are asked to rank them by popularity and quality of contributors. The model utilizes Spearman's rank correlation coefficient to deduce which combination of metrics and weights best correlate to the surveyed developers' ranking in each aspect. We then evaluate potential similarities by comparing the resulting metrics and weights to the reported metrics of the survey. Results are presented and discussed in the final chapters.

This paper is structured into the following chapters, in order:

1. Introduction - General intro of the subject.
2. Background - Related context and literature.
3. Research questions - The research questions are presented.
4. Methodology - Overview of the approach taken to answer those questions.
5. Subjective data collection - Data collected from the developer survey is presented and discussed.
6. Objective data collection - Implementation details of the GitHub data collecting software and definition of metrics used.
7. Analysis - Comparison of the subjective and objective data.
8. Discussion - Findings and threats to validity are discussed.
9. Conclusion - Paper is summarized and concluded.

Chapter 2

Background

This chapter provides the reader with the necessary background for the following chapters. Following sections inform on OSS, Git, GitHub, models currently used to evaluate open-source software, and other related works.

2.1 Spearman's and Pearson's correlation coefficients

Correlation coefficients are heavily used in this paper and it is useful to cover the application and differences of the two used in upcoming chapters, namely Spearman's rank correlation coefficient and Pearson's correlation coefficient. Pearsons's correlation coefficient is a measure of the linear correlation of two sets of data, while Spearman's correlation coefficient measures the degree to which the relationship between two sets of data can be described using a monotonic function [17]. Spearman's coefficient is calculated by calculating the Pearson correlation of the *rank values* of the data sets, meaning a set of values {13,2,7} would have the ranks {3,1,2} in the same order. In essence, this means that the Pearson correlation between two sets of data could be closer to 0 if the relationship is non-linear, but the Spearman correlation between the same sets of data could be high as long as they rank similarly.

Since the method used in this paper mainly revolves around ranking of data, Spearman' coefficient is the main tool for measuring correlation in following chapters. Pearson correlation is utilized in cases where ranking is irrelevant.

2.2 Open Source

The term "open source" emerged in the late 1990's with the birth of the Open Source Initiative (OSI), with its underlying philosophy tracing back to the beginning of software development itself [9]. One of OSI's fundamental interests is facilitating developer collaboration by allowing software and its source code to be freely used, modified and shared. To this end, the Open Source Definition (OSD) was created as a guideline which software licenses must comply with in order to be certified as an OSI approved license [13]. Software with such a license is thereby labeled as open source.

Two of the most popular OSI approved licenses are the GNU General Public License (GPL) and the MIT license (as in Massachusetts Institute of Technology) [8, 12]. These two licenses both adhere to the OSD but have very different legal and practical implications. GPL is a so called *copyleft* license which aims to restrict derived works from becoming proprietary (and therefore not open-source). In contrast, the MIT license is a permissive license which places no such restrictions on derived works. In simplified terms, MIT licensed software can be used freely in proprietary software without placing it under open source requirements while GPL licensed software cannot. With many other open source licenses having different positions on the copyleft-permissive scale, the license plays an important role when evaluating the potential use of an open source component in your software.

2.2.1 Free/libre versus open source

Open source is often included in an acronym, FLOSS, short for "Free/Libre Open Source Software". This is an umbrella term for two types of software with arguably minuscule practical differences [6]. Free/Libre[1] software puts emphasis on the freedom to use the program and code, including distributing copies of modified or unmodified code. The name open source is meant to reflect collaboration and community-driven development.

The acronym for open source software (OSS) is used exclusively in this paper, not because the software discussed is never classified as Free/Libre, but because the collaborative nature of open source better aligns with the context of discussed topics.

2.3 Git

Git is a version control system (VCS) that provides the tools needed for developers to track changes made in their code [18]. This enables software communities to efficiently collaborate on projects and organize their code. The directory which git is initialized in is called a "repository" and usually contains the source code and information necessary for contribution to the project it represents. Developers submit new code to the repository through "commits" which contain the changes made to the code with respect to the previous commit of affected code lines. Different development states can also be separated into "branches" with each branch tracking its own history of commits. A branch can in turn be merged into another

[1] *Free* does not imply free of charge. *Libre* is meant to emphasize this since it roughly translates to "state of liberty" in many Romance languages

branch to reflect the changes made in each branch respectively. Most Git repositories has a main/master branch which contains the current development state of the source code, while other branches contain work in progress. Finally, the local repositories existing individually on each developer's machine are separate from the remote repository called "origin" which is usually hosted by a distributed version control provider such as GitHub. Local and remote repositories are synchronized by processes called "pushing" and "pulling".

2.4 GitHub

In its core, GitHub is a distributed version control provider where you can store your Git version controlled code, but has since its launch in 2008 become much more than this [7]. As organizations have transitioned into open source, developers have recognized the importance of contributing to open source projects from a career standpoint. Developers active on GitHub can refer to their profile in their CV:s which neatly displays the projects and technologies they are familiar with. Furthermore, the collaborative environment of GitHub allows developers to expand their professional networks. Not unlike some social media platforms, GitHub users can "follow" fellow developers and be kept updated on their activity on the platform.

As of writing, GitHub hosts over 200 million repositories, with over 65 million users, and is often the choice of remote Git repository hosting for both hobby projects and large corporations alike.

In the context of this thesis, GitHub repositories share a bit of common terminology that is important to clarify:

- **Stars:** A user can "star" a repository. This action signifies interest from a user and increments the star count for the repository. A user who stars a repository does not receive notifications or news about the repository. It is mostly like a bookmark which is publicly visible to visitors of the user's profile. A "stargazer" is a term used for a specific user who has stared a specific repository.
- **Watchers:** Like the stars feature, when a user "watches" a repository it increments the watchers count for the repository. In addition, watching a repository notifies the user of activity on the repository.
- **Forks:** Users can chose to fork a repository which creates a copy of the code at that point in time, hosted in its own remote repository on GitHub. Any changes made to the source repository is not reflected in the forked repository and vice versa. The number of forks for a repository is simply the total number of forked repositories made at any point in time.
- **Contributors:** A contributor is quite broadly defined by GitHub as someone "who has contributed something back to the project" [1]. Generally, this can mean a person who creates pull requests, posts issues, or carries out code reviews. In the metrics used in this thesis, the term "contributor" is limited to users who contribute code by creating pull requests. The term "developer" is often used as a synonym.
- **Issues:** One of the main ways developers interact with each other on GitHub is through

issues. An issue is written by a user, and despite its name, does not necessarily describe an anomaly in the project such as a bug or an error. It could also be ideas for new features or improvement/refactoring of code. Similar to a thread in an online forum, other users can comment on an issue.

- **Pull requests:** A pull request is a change made to the repository by a contributor which needs to be reviewed and approved by a maintainer in order to be "pulled" into the main branch of the repository. The state of a pull request can either be open, closed, or merged depending on if it is waiting for review, has been declined by a maintainer, or has been merged into the main branch.
- **Maintainer:** A maintainer is a key person within the project that has the authority to merge and approve pull requests to the main branch. Many projects have multiple maintainers and these are collectively referred to as the core team in this paper.

2.5 OSS selection in practice and literature

A number of instruments have been created to help navigate the jungle of available OSS components, either comparatively or individually. This section will give an overview of the main methods that existing assessment models, tools, and services employ to aid in OSS selection.

2.5.1 Quality models

There exists an abundance of models that have been created for facilitating the selection process of OSS components, and comparisons between different quality models have been researched extensively in literature [16, 19, 24, 22]. The majority of models follow a similar process where candidates are identified by following a hierarchical structure to estimate proposed quality characteristics based on metrics. They differ mainly in the degree to which they are either rigid or customizable. Both sides of this spectrum have their shortcomings, since rigid models lack the ability to adapt to the practitioner's needs, and customizable models tend to be very time-consuming in their application.

Haaland et. al. presents certain differences in quality models that allows them to be categorized into three generations [20]:

- Traditional software quality models: Unsurprisingly, mostly revolves around assessing the quality of the product itself by evaluating factors such as maintainability, reliability, efficiency and usability to name a few.
- First generation OSS quality models: With a focus on OSS, these models add to the traditional ones by including community aspects of the assessed software.
- Second generation OSS quality models: Utilizes aforementioned methodologies, but provides tool support to automate parts of the evaluation process.

A trend noticed by Haaland et. al. is that quality models increased in level of automation and number of metrics used. However, the paper is from 2010 and since then there has

been a noticeable absence of newer quality models in literature. One possible explanation for this is that established OSS quality models have seen little or no adoption [21]. Software developers in search of OSS components often pick the first alternative that suits their needs and requirements, rather than spending the time and resources to find the *best* alternative. Experience and opinion of colleagues also seem to play a role. Furthermore, the constraints specific to the situation are usually more important than the criteria proposed by a quality model.

Additional challenges of using quality models include the abundance of alternatives and the lack of time to evaluate them all [26]. This aligns well with the increased granularity of components in today's software. Synopsys, a software management firm writing yearly reports on open source security and risks, stated in their 2021 edition that the average number of open source components used by the audited repositories was 528, which has almost doubled compared to their 2018 report [27]. Applying OSS quality models in this context would not be practical from a business standpoint, as this analysis would require a considerable amount of resources to do for hundreds of OSS components with potentially dozens of alternatives each.

2.5.2 Tools and services

Aside from the numerous quality models established in literature, there exist a few notable practical examples of tools and services that assist practitioners in selecting OSS.

OSS PESTO

One of the more recent contributions to literature regarding OSS selection is OSS PESTO [23], a selection tool with support for collecting GitHub project data and presenting metrics and factors selected by the user. It aims towards providing the means to apply factors and metrics defined by a model of choice, as well as any additional metrics, to real project data. In its current state, the problem regarding customizability discussed in Section 2.5.1 is also relevant here. Setting up the specific parameters for each specific case of OSS selection could be considered tedious by users. It also fails to draw any conclusion about the result (such as a ranking of alternatives). Furthermore, the tool is currently not hosted on a publicly accessible website, which hinders adoption.

Open hub

Maintained by Synopsys, Open hub is a web tool for evaluating and comparing OSS projects existing on various platforms on the internet [14]. Each project is summarized and accompanied with a wide range of metrics such as security vulnerabilities per version, lines of code over time, commits over time, contributors over time, programming languages used, and more.

CHAOSS

CHAOSS, short for "Community Health Analytics Open Source Software", is a Linux Foundation project which defines implementation-agnostic metrics to measure health within the

open source space [4]. The definition of metrics are complemented with data collection strategies where data sources are suggested. These are usually in the form of either technical sources such as project/repository data and source code, or in the form of interview templates and questions that can be applied when collecting data about the organization behind a specific project.

The CHAOSS community has developed a few open source tools revolving around data collection and analysis of their defined metrics [3, 2]:

- **GrimoireLab:** Best described as a collection of tools that can be used for OSS analytics. These tools serve specific purposes such as data retrieval, enrichment, and visualization, and may be used individually or as a complete system. The data is gathered from a multitude of sources such as GitHub, Jira, Bugzilla, Confluence and StackOverflow. Most components of GrimoireLab seem to be actively developed as of writing and multiple projects and services has been built on top of its technology.
- **Augur:** Augur collects data and measures project metrics regarding commits, contributors, issues, pull requests, and more. Unlike GrimoireLab, it focuses on analyzing data available on GitHub, i.e the source code and its GitHub community. It is actively developed.
- **Cregit:** A framework for facilitating analysis and visualization of source code evolution[2]. Inactive since 2019.
- **Prospector:** A tool for aggregating data of open source projects to facilitate unbiased analysis by decision-makers. Initiated by Redhat in 2012 and became part of CHAOSS in 2017 but has been inactive since 2019.

The focus of CHAOSS's actively developed projects, GrimoireLab and Augur, seems to be informing OSS stakeholders and providing versatile means of analyzing relevant data. There is no intention of drawing an absolute opinion based on an OSS project's data. In the FAQ section of the CHAOSS whitepaper, it is stated that a single index for project health is not something they strive to develop due to the many differences between OSS projects, and that they instead focus on defining metrics to aid in forming personal opinion [5].

[2]Source code evolution explains how the source code of a project changes over time, e.g in terms of number of lines, number of changes/commits, or number of authors/contributors.

Chapter 3

Research questions

With respect to the findings discussed in the previous chapter, finding ways to automate parts of the selection process is of increasing relevance to OSS users. Existing tools for evaluating OSS are still in need of human intervention to make sense of the data and make a decision. Current literature and research are yet to explore the possibilities to reduce this human intervention by representing subjectivity with a metric based model. If developer opinion can be represented with such a model, then it points toward an opportunity to automate the ranking of potential OSS candidates with less risk of including wrong or irrelevant metrics.

As a first step into this exploration, we will focus on targeting the answers to a few key research questions, namely:

1. How similar do developers rank projects within the same aspect? (e.g. do developers rank popularity more or less the same?)
2. How similar do developers rank in each aspect for the same project? (e.g. are popularity and contributors ranked similarly?)
3. Which degree of statistically significant correlation can be achieved by comparing the developers' rankings to rankings generated by a weighted metric-based, linear model?
4. What similarities can be found between reported metrics used by the surveyed developers in their evaluation and the resulting metrics chosen by the model?

The following chapter will delve deeper into the method applied to answer these questions.

Chapter 4

Methodology

This chapter discusses the overarching design choices and covers an overview of the method of this thesis. Details and specifics of the two data collections and their analysis (Chapters 5, 6, and 7) are discussed in their respective chapters.

4.1 Debricked

This thesis was carried out at Debricked, a cybersecurity start-up founded in 2018 and situated in Malmö, Sweden. They specialize in open source security and offer SaaS solutions such as tools for dependency vulnerability management and license compliance.

As part of the data science team of Debricked for the duration of the thesis, the author had access to their internal tool for mining GitHub data and was provided with a database of requested project data.

4.2 The projects

Before the chapters regarding data collection and analysis are presented, it is useful to get an overview of the open source projects involved. GitHub was chosen as the common platform for these projects since it is currently the largest development platform and due to its widely accessible API [7]. The metrics of each project were saved onto a read-only database dump which remained static throughout the course of the data collection process. As such, they represent a snapshot of the projects' states the week the developer survey was carried out.

Due to the author's dependence on Debricked's GitHub mining tool, certain parameters had to be put in place to limit the amount of projects and data collected in the mining process.

The GitHub projects shared the following characteristics. They all had:

- Python as their primary programming language (>80%).
- less than 2000 total issues.
- between 15 000 and 26 000 stars.

Python was chosen as the primary language of these projects because the developers who were surveyed primarily used it in their daily work and were accustomed to its ecosystem. When it comes to the number of issues, 2000 was chosen as the cutoff because the size of the projects remain manageable below this point. Above 2000 issues, the amount of data for each project began to surpass what was feasible to collect in terms of both storage and number of API calls. With these two constraints, the top 50 projects with the most stars were selected as candidates. Many of these projects were not proper open-source projects, but rather books, tutorials, or they were not written in English and were therefore excluded. Ultimately, 20 projects remained which were between 15 000 and 26 000 stars. The number of projects was set to 20 in order for the survey to be completed within a reasonable time frame, while still providing enough data for enabling statistical significance.

Table 4.1 is a list of the twenty projects and their respective URL to GitHub for the reader's convenience. Each link follows the convention of https://github.com/<owner>/<name> where owner is the person or organization behind the project, followed by the name of the project. In future references, the shorthand <owner>/<name> format will be used for concision.

For a more detailed overview, Table 4.2 contains some general statistics of each project at the time they were mined from the GitHub API.

4.3 Data collection and analysis

The data collected for this thesis is divided into two categories: subjective and objective. An illustrative flowchart of the process of collecting and analyzing these two sets of data can be seen in figure 4.1

The subjective data is in the form of answers to a survey ("Developer survey" in 4.1) that five developers employed at Debricked took part in on separate occasions within a few days time span. In this survey, each developer is individually asked to rank the aforementioned projects in terms of how popular they think the projects are, and how proficient the contributors of the projects are ("Ranking" in 4.1). Popularity and contributors are referred to as *aspects* throughout this paper. The two aspects were chosen because of the nature of the available data and their fundamental relationship to project community. Stars, forks, contributors, issues, etc. were readily available via the GitHub API and present in all twenty projects. Other potential aspects such as security and code quality were considered but scoped out because of the relatively high complexity of defining and measuring suitable metrics.

Finally, they are asked to list the most important metrics they used in their evaluation, as free text, for each aspect respectively ("Reported metrics" in 4.1). This data is presented in Chapter 5.

Table 4.1: Projects included in the study and ranked by the respondents in the survey.

#	Project
1	https://github.com/chubin/cheat.sh
2	https://github.com/facebookresearch/Detectron
3	https://github.com/localstack/localstack
4	https://github.com/nicolargo/glances
5	https://github.com/apache/airflow
6	https://github.com/tornadoweb/tornado
7	https://github.com/google-research/bert
8	https://github.com/openai/gym
9	https://github.com/donnemartin/interactive-coding-challenges
10	https://github.com/sebastianruder/NLP-progress
11	https://github.com/StevenBlack/hosts
12	https://github.com/CorentinJ/Real-Time-Voice-Cloning
13	https://github.com/matterport/Mask_RCNN
14	https://github.com/magenta/magenta
15	https://github.com/3b1b/manim
16	https://github.com/satwikkansal/wtfpython
17	https://github.com/HelloZeroNet/ZeroNet
18	https://github.com/google/python-fire
19	https://github.com/trailofbits/algo
20	https://github.com/psf/black

Table 4.2: General statistics of the projects listed in table 4.1.

#	Stars	Watchers	Forks	Contributors	Total issues	Total commits
1	19258	468	926	35	138	737
2	23606	998	5045	33	891	140
3	26091	464	1914	274	1786	1432
4	16605	518	1068	109	1248	3855
5	17913	700	6749	1258	1675	9859
6	19378	1053	4915	318	1589	4457
7	24582	962	661	26	970	111
8	21976	1017	5819	245	1180	1220
9	20611	923	3039	42	54	798
10	16547	1282	2699	209	82	676
11	15979	561	1416	92	957	2417
12	19139	550	3620	12	434	268
13	17612	578	7773	41	2107	203
14	15512	816	3087	130	741	1362
15	24803	672	3090	83	689	3039
16	21236	670	1874	52	120	441
17	16001	847	1922	101	2000	3040
18	17609	386	1051	36	183	234
19	19072	444	1559	144	1310	1059
20	17290	166	1028	182	1115	811

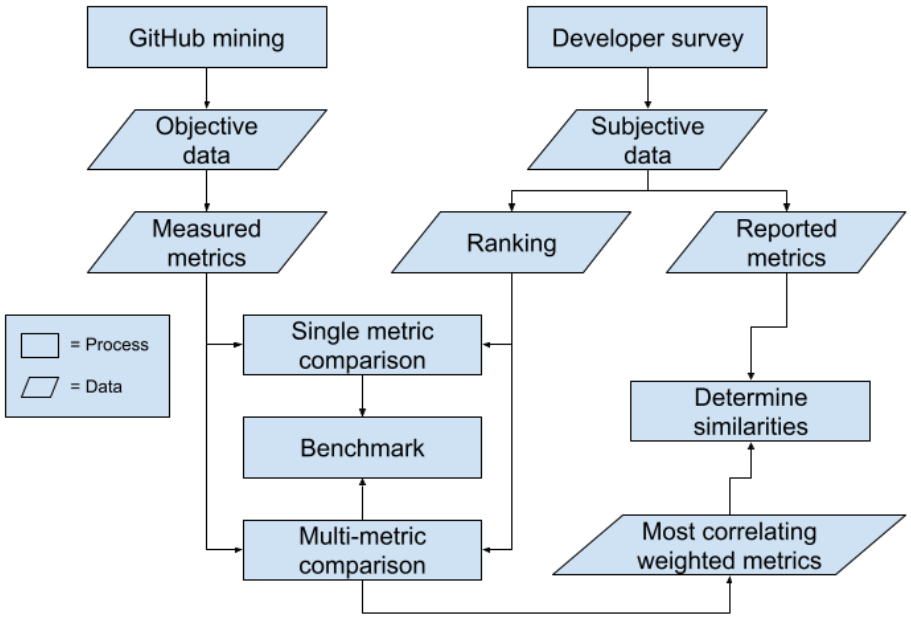


Figure 4.1: Flowchart of the data collecting and analysis process. Arrows indicate chronological order.

The objective data is in the form of measured metrics from each project ("Measured metrics" in 4.1). These metrics are defined with respect to the available data provided by the GitHub API, mostly revolving around the community of each project. Section 2.4 provides an overview of the community elements used to collect data and construct metrics from.

22 metrics were defined in total with the intention of capturing most of the metrics reported by the respondents in the survey. Some metrics takes inspiration from CHAOSS and literature[4]. Ultimately, a large variety of different metrics were included in order to achieve a more diverse analysis. A common denominator for all metrics is that they focus on the current project state, with some metrics taking into account recent history and calculating an average.

The subjective data is subsequently compared to the objective data using Spearman's rank correlation coefficient. At first, one metric at a time ("Single metric comparison" in 4.1) and then multiple metrics ("Multi-metric comparison" in 4.1) where the combination of metrics used and their respective weight in the final scoring is determined by a Python script written by the author. The script was implemented mainly using the libraries NumPy and Pandas. In order to determine the increase in correlation resulting from the multi-metric comparison, it is compared against the single metric comparison ("Benchmark" in 4.1).

By systematically trying out several thousand combinations of metrics, and for each combination testing thousands of random weights, we find the combination of metrics and their weights that results in the highest statistically significant Spearman coefficient ("Most correlating weighted metrics" in 4.1). Finally, these resulting metrics and the order implicated by their weights are compared to the respondents' reported metrics in order to confirm or dismiss any resemblance ("Determine similarities" in 4.1).

Spearman's rank correlation coefficient is used in this paper where the correlation between

two rankings needs to be calculated. This gives us a suitable unit of measurement for determining how similar two rankings are, and the accompanying p-value enables us to confirm or dismiss statistical significance. Pearson's correlation coefficient, the most widely used formula to measure the strength of the relationship between two variables, is also used but to a much lesser extent. Therefore, Spearman's is implied whenever the term "correlation coefficent" is mentioned without specifying otherwise.

4.4 Scope and delimitations

It is important to note that this paper does *not* propose another assessment model for OSS. Rather, it is an experimental study to explore if subjective assessment of OSS can be represented by a naive model. Necessary delimitations to retain this scope has thereby been established:

- The number of aspects explored was bound to two (popularity and contributors) in order to reduce the complexity of the data collection and analysis, while still providing variety in the opinion and metrics used by the surveyed developers.
- Likewise, the number of developers surveyed was limited to five to limit complexity.
- Only projects from GitHub and the data available via the GitHub API has been used.
- The mathematical definition of the model and its components such as normalization, distributions, and weights, was kept simple. That is, normalization was done linearly, and weights were randomized on a uniform distribution.

Chapter 5

Collection of subjective data

The subjective data set was collected by surveying five different developers working for Debricked, all accustomed to the open-source space and with experience in evaluating open-source projects. They were individually instructed to rank 20 different repositories on GitHub based on two aspects:

1. Popularity - How popular is the repository?
2. Contributors - What is the quality of its contributors?

These aspects were chosen for two main reasons.

Firstly, they represent arguably positive traits of OSS with respect to selection. Adopters prefer the OSS component they choose to be popular rather than unpopular, and they prefer the contributors of the project to be proficient rather than inactive and uncoordinated.

Secondly, because of data availability. Most of the data was collected via the GitHub API which due to the social nature of GitHub revolves around the users and their contributions to a large extent.

Information was intentionally left out of the formulation of the survey questions as well. Since the purpose of the survey was to replicate and capture the human evaluation of open source and its subsequent subjectivity, the respondents were free to choose their own strategy for ranking the repositories. This means that they could use whichever information they came across at their own discretion, and were not necessarily limited to only look at GitHub.

The survey itself was made using Google Forms and the ranking process was in the shape of one multi-column question for each aspect, contributors and popularity. Each row had a link to the project on GitHub and the columns had radio-buttons labeled 1 to 20[1]. The

[1]The respondents were instructed before participating that a rank of 1 represented the best repository in

respondents were recommended to keep track of their evaluation separately, in a spreadsheet for example, since the format of the question was not particularly suitable for frequent re-ordering of the rankings.

These are the formulation of the questions in the survey in verbatim:

- "Rank popularity"
- "Rank the contributors"
- "What metrics were important to you when determining popularity? List them from most important to least important."
- "What metrics were important to you when determining quality of contributors? List them from most important to least important."

The two latter questions are referred to as the free text part of the survey.

After ranking the repositories with respect to contributors and popularity, the respondents moved on to the free text section of the survey. This is where they were asked to list the most important metrics they took into consideration during their assessment, ordered from most to least important.

All in all, the survey took approximately four hours for the respondents to complete, with breaks included.

5.1 Result

Table 5.1 shows the result of the rankings made by the respondents which are labeled **A** through **E**. Tables 5.2 and 5.3 contain the answers to the free text question.

5.2 Analysis

This section will present the analysis and insights gathered from the data presented in the previous section. There are two particular questions of interest that will be answered here and those are:

1. How much do the rankings differ between the aspects for the same respondent?
2. How much do the rankings differ between the respondents?

The first question's answer will show us how similar the respondents rank the two aspects. If there is no difference in ranking, i.e popularity and contributors are ranked the same by a certain respondent, then the same metrics are probably used in both assessments and the free text answer should reflect this. In the opposite case, if the aspect-wise ranking is very different then it points toward different priorities of metrics used by the respondent in their assessment.

that aspect, and a rank of 20 represented the worst.

Table 5.1: Ranking results of each aspect made by the respondents. Lower rank means that the respondent perceives the project as better in that aspect.

Respondent	Popularity					Contributors				
	A	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**
chubin/cheat.sh	12	14	14	12	12	13	6	17	9	18
facebookresearch/Detectron	20	18	4	6	4	20	7	3	19	2
localstack/localstack	2	1	6	7	6	2	10	7	3	16
nicolargo/glances	3	12	12	14	19	3	14	11	5	17
apache/airflow	1	4	3	8	2	1	3	1	1	3
tornadoweb/tornado	13	3	7	1	5	10	4	8	6	14
google-research/bert	15	9	2	4	1	15	19	6	18	4
openai/gym	5	2	8	3	3	11	12	2	8	1
donnemartin/interactive-coding-challenges	18	19	13	9	13	19	1	13	17	15
sebastianruder/NLP-progress	11	20	19	19	15	12	13	14	15	5
StevenBlack/hosts	9	10	20	20	18	4	2	19	4	9
CorentinJ/Real-Time-Voice-Cloning	16	15	17	13	9	14	15	18	13	20
matterport/Mask_RCNN	17	17	1	16	8	16	20	5	20	19
magenta/magenta	10	11	11	17	7	6	9	4	2	6
3b1b/manim	7	8	5	18	10	8	16	15	12	7
satwikkansal/wtfpython	19	16	16	11	20	18	17	20	16	10
HelloZeroNet/ZeroNet	8	7	10	10	17	5	18	16	10	13
google/python-fire	14	13	18	5	14	17	8	9	14	11
trailofbits/algo	6	6	15	15	16	9	11	12	11	12
psf/black	4	5	9	2	11	7	5	10	7	8

The answer to the second question will give us an estimate of any subjectivity present between the respondents. If all respondents made the exact same rankings then there would be no sign of subjectivity in the data. Likewise, a very varied ranking between the respondents would tell us that there is subjectivity at play.

Figure 5.1 aims to visualize the answer to the first question. It shows a scatter plot where each data point is a project-respondent pair and its position corresponds to its ranking in the two aspects made by the respondent. With the same example as previously mentioned, if there would be no difference in the way the two aspects were ranked, then the scatters would form a diagonal line from (0,0) to (20,20). This is obviously not the case; the data points yield a Pearson correlation coefficient of 0.359 which indicates that different metrics and priorities of metrics are used by the respondents. Intuitively, one could reason that a popular project recruits better contributors and vice versa, proficient contributors make a project more popular. The modest correlation present could be a product of this, but a significantly larger set of data is required to draw such a conclusion with any degree of certainty.

Figures 5.2 and 5.3 show us the rankings for popularity and contributors respectively, sorted by the arithmetic mean rank of each project. With respect to our second question, it is clear that the respondents are not unanimous in their rankings. Tables 5.4 and 5.5 reinforces this statement. These tables present the Spearman rank correlation matrices for each aspect,

Table 5.2: Free text answers for the popularity aspect.

Respondent	What metrics were important to you when determining popularity? List them from most important to least important.
A	* Recent issues * Stars * Forks
B	* Longlivity - If the repo has survived for long time and if there is a "main guy" * Activity - How has the recent activity been in the project. (Checking contributors activeness) * Community - Community rated by OpenHub * If there are more than one person active in the project. * If recent trends point towards the project's dying. * Active issues and PRs from the recent week (checking pulse) * Issues - Amount of issues * Used by - Amount of people having the project marked as used by * Backed by famous company - A famous company behind the project * Personal bias - If I knew about them before and if the project seemed interesting
C	* Issues * Forks * Stars * PRs * "seriousness" (if it's actually useful or just something cool) * Owner (recognized organization or not) * License (permissive as more popular)
D	* Stars * Forks * Watchers * Downloads
E	* Google Trends (this was sometimes difficult because terms are sensitive) * Watchers * Stars * Forks * Personal exposure (bias towards repositories I knew about, which I think would also naturally be more popular)

which carries out pairwise calculations of how much two respondents' rankings correlate to each other (hence the symmetry).

Interestingly, the similarities seen in the free text answers is not clearly reflected in these correlation coefficients. For example, in the popularity aspect the respondents reported using stars, watchers, and forks to a large extent. Nonetheless the mean value of the correlation coefficients for popularity is approximately 0.3639. Even when looking at the pairs of A-B and C-E where the correlation is over 0.74, there is nothing conclusive to support this number

Table 5.3: Free text answers for the contributors aspect.

Respondent	What metrics were important to you when determining quality of contributors? List them from most important to least important.
A	* Number of commits * Number of high volume contributors * Language in responding to issues Made exceptions for mature projects.
B	* How active the most active contributors are in general * How many followers they have * If they are associated with a famous company * How many popular projects they are active in (I assume if they seem to be a factor to the project success, I rate a repo higher even if the repo seems to be dying and the developer has abandoned it) * If they have been a developer of the project for a long time * No activity the recent 6months get penalized hard
C	* Numbers of contributors with many commits * Recognized organization/company of the developers * "seriousness" (if it's actually useful or just something cool) * Popularity of the project
D	* Recent activity * Is the maintainer active? * Did only one person do it all? * Company backed * General feel of main contributor (active in other projects?)
E	* Semi-recent code commits and other contributions * Existence of and quality of Contribution policy * Merge/reject rate of pull requests * Extent of allowed contributions (some only allow bug fixes and documentation, while keeping the sensitive work to their smaller elite force, which I deemed positive) * Professionalism (some PRs and issues had weird people swearing at developers etc.. This should be moderated. Lack of moderation may be a sign of lack of contributor quality)

in the corresponding free text answers. However, considering the ambiguity in the answers to the free text questions, it proves difficult to explain the correlations (or lack thereof) that do exist. The primary takeaway is that subjectivity is indeed present in this set of subjective data. On a final note, the mean correlation coefficient of the contributors aspect approximates to 0.2573 and the aforementioned reasoning regarding the popularity aspect is applicable here as well.

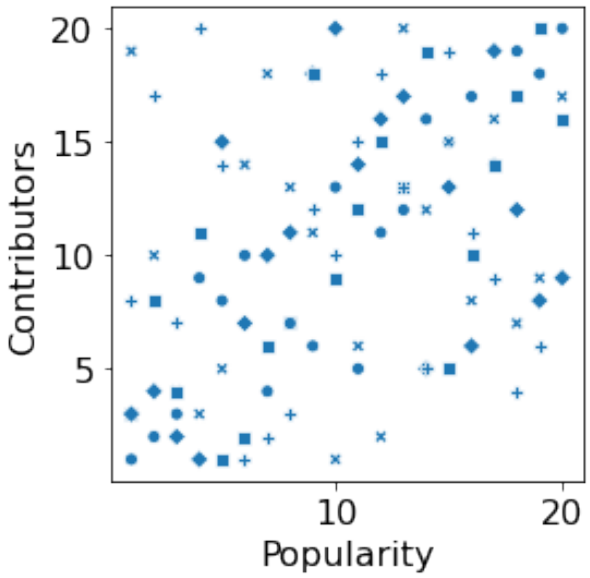


Figure 5.1: A scatter plot of all rankings. Different scatter shapes are different respondents. The x value represents what the respondent ranked that project in terms of popularity, and the y value represents the rank in terms of contributors.

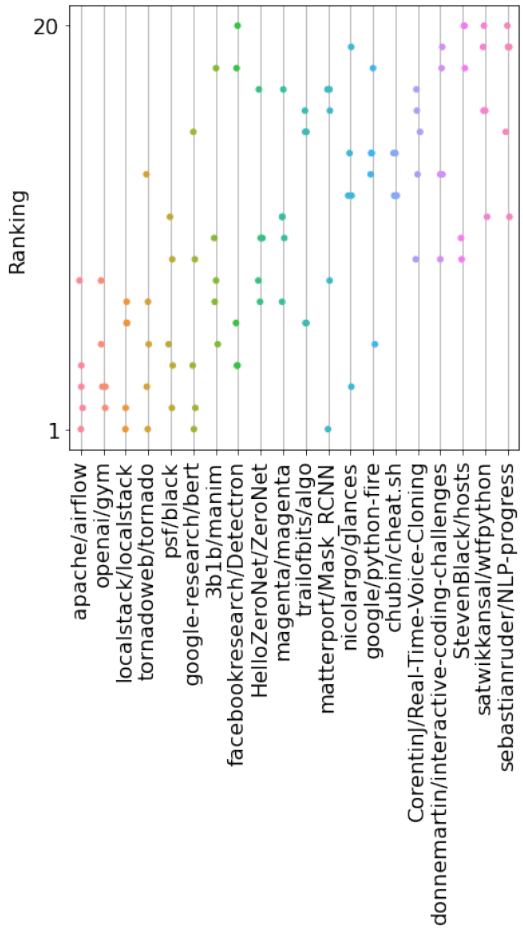


Figure 5.2: The popularity rankings.

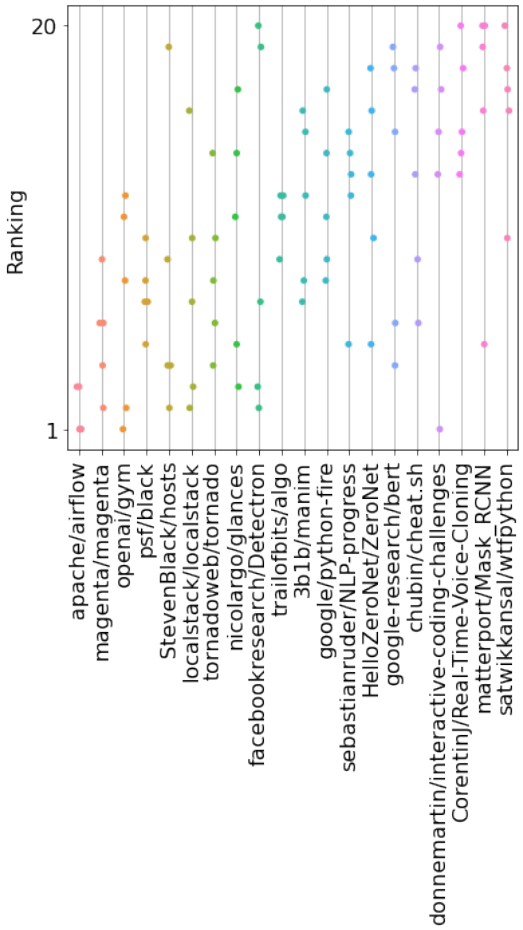


Figure 5.3: The contributors rankings.

Table 5.4: Spearman rank correlation between the respondents popularity ranking.

Respondent	A	B	C	D	E
A	1.000000	0.742857	0.111278	-0.007519	0.027068
B	0.742857	1.000000	0.353383	0.425564	0.351880
C	0.111278	0.353383	1.000000	0.392481	0.741353
D	-0.007519	0.425564	0.392481	1.000000	0.500752
E	0.027068	0.351880	0.741353	0.500752	1.000000

Table 5.5: Spearman rank correlation between the respondents contributors ranking.

Respondent	A	B	C	D	E
A	1.000000	0.144361	0.081203	0.879699	0.028571
B	0.144361	1.000000	0.114286	0.439098	0.114286
C	0.081203	0.114286	1.000000	0.136842	0.448120
D	0.879699	0.439098	0.136842	1.000000	0.043609
E	0.028571	0.114286	0.448120	0.043609	1.000000

Chapter 6

Collection of objective data

This chapter covers the process of collecting the objective data of the twenty repositories presented in 4.2.

6.1 GitHub data

In order to implement metrics and measure data from GitHub repositories, one could call the GitHub API directly and immediately process the data. However, this is impractical for several reasons. It requires a consistent internet connection, it causes considerably longer execution time, and most importantly, the number of API calls per hour is limited. Therefore it is better to store the data gathered from API calls in a local database. This also allows for a more consistent comparison against the subjective data since the metrics can be measured and persisted at the same point in time as the developers carried out their evaluation.

Debricked constructed and provided the author with such a database shortly after the respondents were surveyed. The database contained the raw data needed to implement each metric for each repository. The structure of this database is illustrated in Appendix A.1.

6.2 Metrics

The metrics were defined with inspiration from literature, the CHAOSS initiative, and the reported metrics in the free text answers from the developer survey. The delimitation of only using data available on GitHub dismissed the possibility to use any external metrics but did not prove to be a significant impediment.

Another delimitation made for the sake of simplicity is that only positive metrics are included. That is, each metric fulfills a hypothesis that it would prove useful in the context of open source health: if measured metric 'x' increases, then so does the 'y' aspect of the project". One example of such a hypothesis is the "Total contributors" metric. This metric is calculated as the total number of unique developers on GitHub who has their name in the commits history of the main branch. An increase of this metric would then hypothetically result in an increase in the contributors aspect of open source health. A hypothesis for a metric is always backed by an argument to underline its plausibility. In this case, said argument would be that more total contributors is better because it means more people have inspected the code and bugs are less likely to remain compared to if only a few developers had contributed to the project.

In total 22 metrics were defined and chosen for comparison against the subjective data. A full descriptive list is found in the following subsection, where each metric is documented with a short description, a motivation for its usefulness, and how it is calculated.

It is important to note that no single metric is expected to correlate universally with the subjective data. The simplistic approach to the metric design process is meant to favor quantity with the purpose of maximizing the chance of matching *any* of the metrics to the surveyed developers' opinion. Limiting this collection to a handful of carefully defined metrics would result in a less diverse comparison.

6.2.1 List of metrics

The following metrics were defined for this thesis. Each metric is documented with a description, why it is useful, and how it is calculated.

Recent core team commits

A metric to measure the number of recent commits on the main branch, created by core team members. **This is useful because** the core developers' activity is key for progress. Measuring their commits is a good indicator of how involved they are in the project. **Calculated as** the number of commits created by users who can merge pull requests, the past 10 weeks.

Contributor influence

A metric to measure the influence the contributors have, meaning how much attention they draw to the project. **This is useful because** influential contributors highlights the repository to their followers on Github by notifying them whenever they make a pull request to that repository, which in turn could lead to an increase in contributors and the overall community. **Calculated as** the number of pull requests a certain developer has made the past 10 weeks, multiplied with their follower count and summing over all developers.

Total stargazers

A metric to measure the total amount of stars the repository has. **This is useful because** Github users often star a repository to bookmark it as interesting. Thus, by looking at the

total amount of stars one could estimate the accumulated interest a repository has. **Calculated as** the total star count.

Core team issue closing

A metric to measure the number of issues the core team has recently closed. **This is useful because** the core developers' activity is key for progress. Measuring their responsiveness to issues is a good indicator of how involved they are in the project. **Calculated as** the number of issues closed by users who can merge pull requests, the past 10 weeks.

Company involvement

A metric to measure how much involvement in the project there is from a company. **This is useful because** projects with a clear representation of committers from the same company might indicate that said company is investing resources into the project. **Calculated as** the highest ratio of unique committers from the same company the past 21 weeks.

Developer velocity

A metric to measure how much code the average developer produces. **This is useful because** too much lines of code added per week may result in flawed and buggy software, while too little could be a sign of inactivity. **Calculated as** the total diff of commits (additions - deletions) each week divided by the number of developers of those commits, averaged between the past 10 weeks.

Loyal developer commits

A metric to measure the number of commits loyal (long-term) contributors have recently made. **This is useful because** it tells us if long-term contributors are still active or have moved on to other projects. **Calculated as** the number of commits past 21 weeks, made by contributors who have made at least 50 commits before the lookback period.

New contributors

A metric to measure the number of new contributors, who have at least one merged pull request. **This is useful because** new contributors is a sign that the project is attractive to developers and leads to a more diverse community with more eyes on the code base. **Calculated as** the number of contributors who had their first merged pull request the past 52 weeks.

Recent pull requests

A metric to measure the number of recent pull requests made. **This is useful because** pull requests is the standard way for developers to contribute to a repository. Thus, measuring the number of pull requests made is a good indicator of how active the contributors are. **Calculated as** the number of pull requests created the past 10 weeks.

Total watchers

A metric to measure the total amount of watchers the repository has. **This is useful because** when a Github user watch a repository, they get notifications about its activity. Since most users are unlikely to watch repositories they aren't interested in, the total watchers count is a good indicator of the current interest of a repository. **Calculated as** the total watchers count.

Recent issues

A metric to measure the number of recently posted issues. **This is useful because** issues are a sign of an active community. A lot of issues does not necessarily mean a defect software, but rather that the users are engaged in improving the project. **Calculated as** the number of issues created the past 10 weeks.

Developers per commit

A metric to measure the arithmetic mean of developers per commit over the lifetime of the project. **This is useful because** a healthy and diverse project should have many developers looking over the code base which is good for collective ownership and a sign of maintainability. **Calculated as** the total number of contributors divided by the total number of commits.

Closed issues per developer

A metric to measure the number of issues the developers have recently closed. **This is useful because** it gauges how responsive the developers are to its users' feedback. **Calculated as** the number of closed issues past 52 weeks, divided by the number of unique developers who closed those issues.

Recent merges

A metric to measure the number of recently merged pull requests. **This is useful because** normally, the core team members are the only developers who can merge pull requests. Without anyone merging pull requests, no changes can be made to the main branch. This is why it is important to keep track of the number of recent merges made. **Calculated as** the number of pull requests merged past 10 weeks.

Recent commits

A metric to measure the number of recent commits present on the main branch. **This is useful because** commits play a central role in estimating the activity of a project. Lack of recent commits to the main branch may be a symptom of an inactive community, absent maintainers, deprecation, etc. **Calculated as** the number of commits in the main branch created in the past 10 weeks.

Contribution skew

A metric to measure the skewness contributions between the developers, meaning how the volume of pull requests are divided between the contributors. A highly positive skew means that the pull requests are evenly distributed between the contributors, while a highly negative skew means that a handful of contributors are responsible for most of the pull requests. **This is useful because** a high number of contributors is good for many reasons, but may be misinterpreted if a few contributors are doing all the work. The contribution skew metric is a practical method of validating if the contributions are productively distributed. **Calculated as** the statistical skewness of the number of pull requests each contributor made.

External pull requests

A metric to measure the arithmetic mean of the contributors' merged pull requests made to other repositories. **This is useful because** an accepted pull request is usually a sign that the contribution is deemed useful to a repository. By measuring the average number of merged pull requests the contributors have in other repositories, you could estimate the overall experience the average contributor has. However, this metric does not take into consideration the relevance or age that experience. **Calculated as** the average number of merged pull requests the contributors has made to other repositories.

Total contributors

A metric to measure the total amount of contributors of the repository. **This is useful because** a high number of total contributors is a sign of a diverse community. **Calculated as** the number of unique developers in the main branch commit history.

Developer lifetime

A metric to measure how long the developers, on average, are actively contributing to the project. **This is useful because** two developers who have contributed actively for ten months have probably done more for the project than ten developers who have contributed to the project for two months. **Calculated as** the consecutive weeks of activity of a moving window, averaged between all contributors. The window has a length of 8 weeks where the contributor is marked as active if at least 3 weeks of that window includes at least one commit.

Total forks

A metric to measure the total amount repositories that has been forked from this repository. **This is useful because** a high fork count is a sign of interest from the developer community. **Calculated as** the total fork count.

Pull requests merged per developer

A metric to measure the arithmetic mean of merged pull requests per developer. **This is useful because** it gauges the productivity of the contributors. However, a low score on this metric may also mean that the core team is not merging the pull requests that the contributors are

producing. **Calculated as** the number of pull requests which were merged within the past 52 weeks, divided by the number of distinct developers of those pull requests.

Recently closed issues

A metric to measure the number of recently closed issues. **This is useful because** closed issues is important to measure since it tells you if upcoming bugs, ideas and problems are being addressed by the maintainers. **Calculated as** the number of issues closed the past 10 weeks.

Chapter 7

Analysis

With the two collections of data in place, it was time to make sense of the information. This chapter covers the comparative analysis between the subjective data, described in Chapter 5, and the objective data described in Chapter 6.

This chapter is comprised of two sections. The first section looks at the comparison between each single metric and the subjective data. The second section presents the comparison of multiple metrics to the subjective data.

In both methods, the comparison makes use of Spearman's rank correlation coefficient. This coefficient is bound between -1 and 1 and represents how similar two rankings are. Along with the correlation coefficient itself, the calculation also outputs a p-value which in statistical terms represents the probability that the given correlation happened by chance if the null hypothesis were to be true. In our case, the null hypothesis is as follows:

H0: There is no monotonic association between the respondents' ranking and the ranking derived from the objective data.

In this thesis, we only consider correlation coefficients with p-values below 0.05. This means that there is less than a 5% chance that the calculated correlation coefficient happened spontaneously if there is no monotonic association between the rankings.

7.1 Single metric comparison

In this section, we compare the ranking of a single metric to the subjective data collected.

The purpose of looking at the comparison between a single metric and the responses of the survey is twofold. Firstly, we want to find out if there is a naive evaluation method present

in any of the respondents' ranking, e.g only looking at the number of stars of a project when rating popularity. Secondly, the results will give us a benchmark to compare against in the following section where multiple metrics will be taken into consideration.

The results of the single metric comparison is presented in Table 7.1 and Table 7.2 for popularity and contributors respectively. Each value shows the Spearman coefficient between the ranking made by a respondent and the ranking derived from the metric. For example, the ranking of the "Total Stargazers" metric is simply made by sorting the resulting scores of each project from high to low, where the project with the most stars is on rank one, and the project with the least stars is on rank twenty.

Table 7.1: Popularity: Spearman coefficient between ranking from a single metric and a respondent ranking of the popularity aspect. Bold coefficients have a p-value below 0.05.

	A	B	C	D	E	avg
Recent Issues	**0.54**	**0.57**	0.39	-0.02	0.34	0.364
Recently Closed Issues	**0.51**	**0.62**	0.28	0.05	0.32	0.356
Total Contributors	**0.73**	**0.65**	0.11	0.13	0.10	0.344
Recent Pull Requests	**0.58**	**0.49**	0.08	-0.05	0.33	0.286
Total Forks	-0.22	0.04	**0.67**	0.14	**0.73**	0.272
Total Stargazers	-0.18	0.17	**0.46**	0.42	**0.48**	0.270
Recent Merges	**0.57**	**0.52**	-0.06	0.14	0.16	0.266
Contributor Influence	0.36	0.39	-0.04	0.12	0.22	0.210
New Contributors	**0.75**	**0.52**	-0.12	-0.05	-0.05	0.210
Recent Commits	**0.56**	0.40	-0.16	0.04	0.05	0.178
Developers per Commit	-0.17	-0.09	0.30	0.36	**0.46**	0.172
Core Team Issue Closing	**0.50**	**0.51**	-0.23	0.03	-0.05	0.152
Developer Velocity	0.44	0.20	-0.05	0.04	0.11	0.148
Recent Core Team Commits	**0.53**	0.40	-0.18	-0.05	-0.05	0.130
Loyal Developer Commits	**0.50**	0.34	-0.12	-0.16	-0.03	0.106
Pull Requests Merged	0.40	0.35	-0.18	-0.02	-0.03	0.104
Closed issues per developer	**0.49**	**0.46**	-0.29	0.01	-0.29	0.076
Total Watchers	-0.29	-0.14	0.26	0.09	0.38	0.060
Contribution Skew	-0.11	-0.00	0.22	0.11	-0.04	0.036
Company Involvement	-0.15	-0.03	-0.18	0.00	-0.11	-0.094
External Pull Requests Merged	-0.10	0.02	-0.47	0.11	-0.42	-0.172
Developer Lifetime	-0.05	-0.18	-0.25	-0.44	-0.26	-0.236

As we can see, most computed correlation coefficients around 0 have a p-value larger than 0.05, while correlation coefficients closer to 1 or -1 have p-values below 0.05. This is to be expected since higher correlations (or inverse correlations) are intuitively less likely to happen by chance. That is not to say, however, that higher correlations with low p-values are evidence of the respondents using that specific metric in their evaluation. Only in contrast to the free text answers can such conclusions be discussed, and will be discussed in Chapter 8.

Overall, there is no clear indication that any of the respondents relied on any singular metric

Table 7.2: Contributors: Spearman coefficient between ranking from a single metric and a respondent ranking of the contributors aspect. Bold coefficients have a p-value below 0.05.

	A	B	C	D	E	avg
Total Contributors	**0.68**	0.30	0.32	**0.69**	0.25	0.448
New Contributors	**0.71**	0.21	0.07	**0.72**	0.35	0.412
Loyal Developer Commits	**0.61**	**0.62**	-0.03	**0.80**	-0.12	0.376
Contributor Influence	0.33	**0.72**	0.19	**0.51**	0.08	0.366
Recent Commits	**0.55**	**0.60**	0.00	**0.74**	-0.12	0.354
Recent Pull Requests	**0.55**	0.34	0.15	**0.57**	0.08	0.338
Recent Merges	**0.52**	**0.46**	0.10	**0.64**	-0.08	0.328
Recent Core Team Commits	**0.63**	**0.53**	-0.17	**0.77**	-0.23	0.306
Core Team Issue Closing	**0.60**	0.20	-0.12	**0.73**	-0.13	0.256
Recently Closed Issues	**0.58**	-0.08	0.15	**0.49**	-0.04	0.220
Recent Issues	**0.61**	-0.11	0.21	**0.48**	-0.10	0.218
Closed issues per developer	**0.63**	0.14	-0.25	**0.65**	-0.17	0.200
Developer Velocity	0.31	0.08	0.16	0.42	-0.08	0.178
Pull Requests Merged	0.43	**0.47**	-0.25	**0.49**	-0.32	0.164
Developers per Commit	-0.35	-0.06	**0.57**	-0.36	**0.45**	0.050
Total Watchers	-0.22	-0.07	0.26	-0.22	**0.48**	0.046
Total Forks	-0.17	-0.26	**0.58**	-0.25	0.32	0.044
Company Involvement	-0.13	**0.48**	-0.17	0.05	-0.14	0.018
Developer Lifetime	0.17	0.07	-0.32	0.20	-0.34	-0.044
Contribution Skew	-0.17	-0.17	0.05	-0.25	0.04	-0.100
Total Stargazers	-0.34	-0.12	0.16	-0.31	0.09	-0.104
External Pull Requests Merged	-0.05	0.18	**-0.52**	0.07	**-0.58**	-0.180

when ranking either popularity nor contributors. The largest correlation coefficient with a p-value below 0.05 for each respondent will be used to benchmark the performance of the ensuing results from the multi-metric comparison.

7.2 Multi-metric comparison

The purpose of the multi-metric comparison is to compare the best combination of metrics and weights to the result of each respondent in the survey. If the derived metrics and weights show resemblance to a respondent's free text answer in the survey, then the respondents ranking can be represented by the model with an accuracy proportional to the correlation between the two (given that the p-value suggests statistical significance).

In broad strokes, this is done by finding the linear combination of weights and metrics that yields the highest Spearman coefficient with a p-value below 0.05 for a particular respondent's aspect ranking. As we do not wish to speculate on which metrics are of relevance for each aspect, all metrics will be used as potential candidates for both aspects.

We will begin by introducing and motivating the implementation of the multi-metric com-

parison program, followed by the results and the insights gained from them.

7.2.1 Implementation

Following subsections will sequentially explain how the multi-metric comparison is implemented. The author's implementation used a vectorized approach using the linear algebra library NumPy to increase performance, but with respect to reproducibility it is easier to both explain and understand if it is divided into bite-sized subsections, with each subsequent subsection relying on the data provided by its predecessor.

Normalization

Since we start off with the raw measurements of each metric from the previous chapter, we first need to normalize this data to a common scale. Otherwise, metrics with higher score will be over-represented in the final result, and vice versa for lower scoring metrics. The scale is set to be linear between 1 and 20, similar to the scale used by the respondents in Chapter 5 when ranking the projects, but reversed, and continuous instead of discrete. It is reversed since the normalized score of 20 is assigned to the top scoring project, and 1 is assigned to the lowest scoring project. After the weights have been applied we will reverse the result again to correspond to the respondents ranking. It is continuous since the remaining repositories are assigned real values between 1 and 20. The normalization formula is as follows:

$$s_{norm} = \frac{s_{raw} - \min s_{raw}}{\max\left(s_{raw} - \min s_{raw}\right)} * 19 + 1$$

Where s_{raw} is the raw score vector and s_{norm} is the resulting normalized score vector.

After normalizing all metrics we end up with a 2-dimensional matrix consisting of the normalized scores for each project-metric pair.

Metric combinations

It is not desirable to include all 22 metrics in every weight generation, since we want to validate the result against the respondents free text answers which mostly consist of 4-5 metrics that can be mapped to the defined metrics. We also do not need to include singleton sets, since this analysis was carried out in section 7.1. Therefore, we will create all combinations of 2-4 metrics, without repetition. That gives us

$$\binom{22}{4} + \binom{22}{3} + \binom{22}{2} = 9086$$

different combinations to generate weights for. By comparison, choosing an upper limit of five would result in 35420 metric combinations which takes almost four times longer to execute. Opting for a lower upper limit also allows us to put more resources into generating weights. This balancing act between metric combinations and number of generated weights will be further discussed in the next step.

For each of our 9086 metric combinations, we use the normalized score matrix to construct new matrices, one for each combination which consists of only the project-metric pairs resulting from that combination of metrics.

Weight generation

The next step is to generate the weights that yield the highest correlation when compared to the subjective data. This step and the next should be iterated for each metric combination, since we only need to store the best performing sets of weights for each combination, respondent, and aspect.

Each weight is a real value between 0 and 1, and the sum of weights for a given combination of metrics should add up to 1, such that

$$\{w_m \in \mathbb{R}, 0 < w_m < 1, \sum_{m \in M} w_m = 1\}$$

For a each metric m and its weight w_m in the given set of metrics M.

This means that there are infinite possible sets of weights, so for practical purposes, we will simply randomize the weights on a uniform distribution n times to get n sets of weights $w_m, m \in M$. The chance to find a higher correlation increases as n increases, but improvements are virtually unnoticeable for values above $n = 10000$ so this is the number used in the results presented in Section 7.2.2. This is also why the number of metrics per combination was limited to four. At higher limits, the number of generated weights needed to be lowered in order for the program to be executed within a reasonable time frame, even though a higher correlation could be achieved.

After generating n different sets of weights, we multiply the weights with our normalized scores to get our weighted scores, and sum the weighted scores for each metric to get the final score for each project. What we end up with are n different sets of scores for all twenty projects, for the current combination of metrics in our iteration. Lastly, the scores are converted into rankings in order to compare them to the subjective data. The lowest scoring project gets rank 20, and the highest scoring project gets rank 1.

Finding the best-case weights

The last step is to see which of our n rankings, and thus their associated weights, yield the highest statistically significant correlation when compared to the subjective data. For each of the ten rankings made in the survey, we store the underlying weights of the rankings made by the program that best fit that data, i.e the ones with the highest Spearman coefficient (where $p < 0.05$). This results in ten sets of weights for each metric combination after the iteration is complete.

In the following subsection, the best metric combinations and weights for each aspect-respondent pair will be presented.

7.2.2 Result

Table 7.3 shows the result from the multi-metric comparison with the highest statistically significant correlation compared to the single metric with the highest statistically significant correlation, for popularity and contributors respectively. As mentioned in Section 7.1, the results from the multi-metric comparison would not be very interesting to look at if the correlation coefficient was lower than the "best" single metric for each respondent. In all cases, the multi-metric comparison resulted in a higher Spearman coefficent with the exception of respondent D in the popularity aspect where none of the single metric correlations were statistically significant. Therefore, a comparison to the multi-metric result would be unjustified.

Table 7.3: Spearman correlation coefficients of best single metric versus best multi-metric result, for each respondent.

Popularity		
Resp.	**Single metric**	**Multi-metric**
A	0.75	0.87
B	0.65	0.79
C	0.67	0.82
D	-	0.72
E	0.73	0.92

Contributors		
Resp.	**Single metric**	**Multi-metric**
A	0.71	0.90
B	0.72	0.89
C	0.58	0.81
D	0.80	0.96
E	0.48	0.7

Table 7.4 shows the actual weights and metrics that resulted in the correlation coefficients listed in table 7.3. If the highest correlating metric from the single metric comparison in section 7.1 is present, it is marked in boldface.

By qualitatively comparing this data to the free text answers given by the respondents in Chapter 5, we can determine if there is any resemblance.

Comparison to free text answers

The comparison done per aspect and per respondent. Only a short summary of the author's findings is given for each respondent and should provide enough insight to determine if best-case weights from the multi-metric comparison were accurate or not. This will be discussed in Chapter 8.

Popularity

Respondent A: Reported valuing recent issues, stars and forks. Multi-metric result mostly revolved around contributors. No similarities found.

Respondent B: Reported many metrics used in their evaluation, most of them related to contributors. Arguably some overlap between the respondent's second most important metric "Recent activity" and the second most weighted metric from the multi-metric result "Recent commits". Few similarities found.

Table 7.4: Best case weights and metrics for each respondent for popularity (left) and contributors (right) respectively. Best metric from the single metric comparison are bold.

Resp.	Metric	Weight
A	**New Contributors**	0.68
	Total Contributors	0.15
	Developer Lifetime	0.11
	Developer Velocity	0.05
B	**Total Contributors**	0.45
	Recent Commits	0.24
	Closed issues per developer	0.19
	Total Stargazers	0.12
C	Loyal Developer Commits	0.58
	Developer Velocity	0.30
	Total Forks	0.08
	Total Stargazers	0.04
D	External Pull Requests Merged	0.37
	Total Stargazers	0.26
	Developers per Commit	0.19
	Total Contributors	0.18
E	Developer Velocity	0.74
	Loyal Developer Commits	0.11
	Total Forks	0.11
	External Pull Requests Merged	0.04

Resp.	Metric	Weight
A	Total Contributors	0.49
	Loyal Developer Commits	0.38
	Developer Lifetime	0.07
	Closed issues per developer	0.06
B	Loyal Developer Commits	0.49
	Contributor Influence	0.46
	Company Involvement	0.03
	Developers per Commit	0.02
C	Total Contributors	0.39
	Total Forks	0.34
	Developers per Commit	0.18
	Company Involvement	0.09
D	**Loyal Developer Commits**	0.55
	Recent Merges	0.27
	Total Contributors	0.15
	Developer Lifetime	0.03
E	New Contributors	0.59
	Total Watchers	0.28
	Developers per Commit	0.13

Respondent C: Only similarities found with "Total forks" and "Total Stargazers". However, the generated weights for these metrics are too small to match the reported importance by the respondent. Very few similarities found.

Respondent D: Stars reported as most important and "Total Stargazers" ended up being the second most weighted metric by the program. No other metric matches free text. Very few similarities found.

Respondent E: Barely any resemblance other than to "Total forks" which was weighted very low. Very few similarities found.

Contributors

Respondent A: Reported vague or subjective metrics used in their evaluation. Weak overlap between "Loyal Developer Commits" and reported most important metric "Number of commits". Very few similarities found.

Respondent B: Top three reported metrics can arguably be mapped to the three most weighted metrics from multi-metric result. However, most important reported metric "How active the most active contributors are in general" is not exactly the definition of "Loyal Developer

Commits". Some similarities found.

Respondent C: Overall, reported metrics are vague and difficult to map to defined metrics. Only connection found is between reported metric "Company backed" and defined metric "Company involvement", however the weight is relatively low. Very few similarities found.

Respondent D: Top reported metrics "Recent activity" and "Is the maintainer active?" are related to top weighted metrics "Loyal Developer Commits" and "Recent Merges", though the definition of activity is itself subjective. Third most important reported metric "Did only one person do it all?" is ambiguous but could be tied to "Total Contributors" if the respondent mean more contributors is better. Some similarities found.

Respondent E: Mostly very unspecific/subjective reported metrics in the free text answer which makes it difficult to link to defined metrics. No similarities found.

Chapter 8

Discussion

To summarize the multi-metric comparison, the similarities were largely few and far between. Many of the reported metrics in the free text answers were to some degree implemented into the model as well. Despite this, they were not weighted accordingly by the program. There are many potential explanations for this which we will continue to delve into.

A big difference between the reported metrics in the free text answers and the defined metrics is that the defined metrics, for the sake of simplicity, are very specific as they return a simple scalar/number for a certain project. For example, respondent A reported that "Recent issues" was the most important metric when they were determining the popularity of the projects. A defined metric exists with that exact name, and is defined "as the number of issues created the past 10 weeks". What if the respondent only looked at the past two weeks, or past six months? Maybe the respondent looked at the number of comments posted on the most recent issues as well? Even if the defined metrics were tailored to the respondents reported metrics, a lot of guesswork would be needed to account for all possible variables.

Another interesting design alternative to consider is using other normalization functions than the linear normalization used in this paper. Using a linear normalization function implies that measurements of a certain metric differ in importance linearly between the projects. One could argue that this is not always the case. To give an example, consider the "Total Stargazers" metric. Using linear normalization, the difference in normalized score between two projects with 20 and 3020 stars respectively would be the same as for two projects with 30 000 and 33 000 stars. However, a person assessing these projects might see the latter difference as much less significant than the first. Put simply, measurements are compared differently depending on the metric. Thus, the choice of normalization function is arguably also dependent on the metric.

A constraint that would be useful in hindsight is to set a lower limit for the generated weights

to remove the risk of generating vanishingly small weights. As an example, the metric "Developers per Commit" was given a weight of 0.02 by the multi-metric comparison program for respondent B in the contributors aspect. Intuitively, respondents are not likely to include metrics of such low relative importance in their reported metrics, and the resulting correlation would likely not have suffered.

On a positive side note, the resulting statistically significant Spearman coefficients from the single metric comparison were almost all above zero. This strengthens the hypothesis that the defined metrics indeed measure positive traits of a project, rather than the opposite.

8.1 Threats to validity

The following section will discuss potential threats to internal and external validity. The list of threats summarized by Wohlin et. al will be used as a checklist where applicable [28].

Internal validity reflects how well a study is designed and conducted. The following threats to internal validation are identified:

- The five developers were surveyed within a short enough timeframe for the state of the projects to remain relatively unchanged. However, the specific time each developer carried out their evaluation could have an impact on their results due to factors such as current workload and other distractions.
- Both the design of the survey, the software used for data collection and analysis, and the definition of metrics are prone to human error and oversight.
- The developers selected for the survey may have varied accuracy of representing reality, i.e. how well their evaluation methods in the survey corresponds to the evaluation methods they apply in real life situations.

External validity explains how well the results of the study can be used to represent the real world. The following threats to external validation has been identified:

- The surveyed developers are likely to not be perfectly representative of the larger population of OSS practitioners since they are employees at the same company with similar age and experience.
- The chosen projects were not limited to a certain domain which may impact the set of metrics the respondents used. If all projects were seen as alternatives to the same desired functionality, a different outcome may have been observed in the results.

8.2 Implications

It is important to recognize that even if the linear model would be accurate in representing the survey respondents' opinion, it would unlikely be enough to prove or disprove that the model actually works. A set of five developers is simply not enough to reach a statistically significant conclusion about the performance of the model. On the other hand, the bleak results given by the multi-metric comparison and analysis in this paper does not entirely rule

out the possibility that the model would perform better on a larger set of surveyed developers where outliers could be more easily identified. To this end, the method of deriving a metric based model presented in this paper may prove useful in future work, and the result signals a need for larger sets of data to be used as input.

Chapter 9

Conclusion

The result of this thesis suggests that a simple linear model consisting of 22 community-centered metrics can not sufficiently represent the five developers' subjective evaluation methods, even after an intensive survey as input.

Looking back at the four research questions stated in Chapter 4, we can conclude the following:

- The developers do not seem to rank the projects convincingly similar to each other within the same aspect. The mean Spearman coefficient when pairwise comparing their rankings is 0.364 and 0.257 for popularity and contributors respectively.
- The developers rank the same project differently depending on which aspect is being evaluated. Figure 5.1 visualizes this finding, and the data points yield a Pearson correlation coefficient of 0.359 which is indicative of different metrics being used by the developers when evaluating each aspect.
- The model achieved a mean Spearman correlation of 0.838 when averaging the ten statistically significant values of the multi-metric comparison.
- The metrics reported by the respondents and the metrics resulting from the multi-metric comparison showed very little resemblance in the author's qualitative comparison.

With an exponentially increasing number of options within OSS, and literature describing and comparing the cumbersome models for manually assessing and selecting OSS components, further research into ways of automating this process is warranted. Paradoxically, the reason behind why most assessment models take time and effort to adopt is because of the need for human interaction to account for subjectivity.

Alternative angles to the approach taken in this paper are encouraged, such as:

- Limiting the dataset to domain-specific projects, such as only comparing front-end frameworks such as React, Vue and Angular.
- Exploring larger sets of developer opinion.
- Testing non-linear models and normalization methods.
- Including metrics from other sources such as code quality metrics.

References

[1] Anatomy of an open source project. https://opensource.guide/how-to-contribute/#anatomy-of-an-open-source-project. [Online: accessed 2021-08-08].

[2] CHAOSS github page. https://github.com/chaoss. [Online: accessed 2021-08-08].

[3] CHAOSS software. https://chaoss.community/software/. [Online: accessed 2021-08-08].

[4] CHAOSS web site. https://chaoss.community/. [Online: accessed 2021-08-08].

[5] CHAOSS whitepaper. https://github.com/chaoss/whitepaper/blob/master/whitepaper.md. [Online: accessed 2021-08-08].

[6] FLOSS and FOSS. https://www.gnu.org/philosophy/floss-and-foss.html. [Online: accessed 2021-08-08].

[7] Github about. https://github.com/about. [Online: accessed 2021-08-08].

[8] GNU general public license. https://www.gnu.org/licenses/gpl-3.0.html. [Online: accessed 2021-08-08].

[9] History of the OSI. https://opensource.org/history. [Online: accessed 2021-08-08].

[10] It's an open-source world: 78 percent of companies run open-source software. https://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software/. [Online: accessed 2021-08-08].

[11] Linux kernel github repository. https://github.com/torvalds/linux. [Online: accessed 2021-08-08].

[12] MIT license. `https://opensource.org/licenses/MIT`. [Online: accessed 2021-08-08].

[13] The open source definition. `https://opensource.org/osd`. [Online: accessed 2021-08-08].

[14] Openhub. `https://www.openhub.net/`. [Online: accessed 2021-08-08].

[15] Alain Abran and Rafa Al-Qutaish. ISO 9126: Analysis of quality models and measures. In Alain Abran, editor, *Software Metrics and Software Metrology*, chapter 10, pages pp. 205–228. Wiley-IEEE Computer Society, 2010.

[16] Adewole Adewumi, Sanjay Misra, Nicholas Omoregbe, and Luis Fernandez-Sanz. FOSSES: Framework for open-source software evaluation and selection. *Software: Practice and Experience*, 49, 2019.

[17] Gunnar Blom, Jan Enger, Gunnar Englund, Jan Grandell, and Lars Holst. *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur AB, 2017.

[18] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[19] Umm e Laila, Adnan Zahoor, Khalid Mehboob, and Sarfaraz Natha. Comparison of open source maturity models. *Procedia Computer Science*, 111:348–354, 2017. The 8th International Conference on Advances in Information Technology.

[20] Kirsten Haaland, Ruediger Glott, and Anna Tannenberg. Free/libre open source quality models-a comparison between two approaches. *4th FLOSS International Workshop on Free/Libre/Open Source Software*, 2010.

[21] Øyvind Hauge, Thomas Østerlie, Carl-Fredrik Sørensen, and Marinela Gerea. An empirical study on selection of open source software - preliminary results. In *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 42–47, 2009.

[22] Valentina Lenarduzzi, Davide Taibi, Davide Tosi, Luigi Lavazza, and Sandro Morasca. Open source software evaluation, selection, and adoption: a systematic literature review. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 437–444, 2020.

[23] Xiaozhou Li and Sergio Moreschini. OSS PESTO: An open source software project evaluation and selection tool. In Davide Taibi, Valentina Lenarduzzi, Terhi Kilamo, and Stefano Zacchiroli, editors, *Open Source Systems*, pages 42–50. Springer International Publishing, 2021.

[24] Sanjay Misra, Adewole Adewumi, Nicholas Omoregbe, and Broderick Crawford. A systematic literature review of open source software quality assessment models. *SpringerPlus*, 2016:1936, 2016.

[25] Gregorio Robles, Igor Steinmacher, Paul Adams, and Christoph Treude. Twenty years of open source software: From skepticism to mainstream. *IEEE Software*, 36:12–15, 2019.

[26] Klaas-Jan Stol and Muhammad Ali Babar. Challenges in using open source software in product development: A review of the literature. pages 17–22, 2010.

[27] Synopsys. Open source security and risk analysis report. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html, 2021. [Online: accessed 2021-08-08].

[28] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Springer, 2012.

Appendices

Appendix A

Database

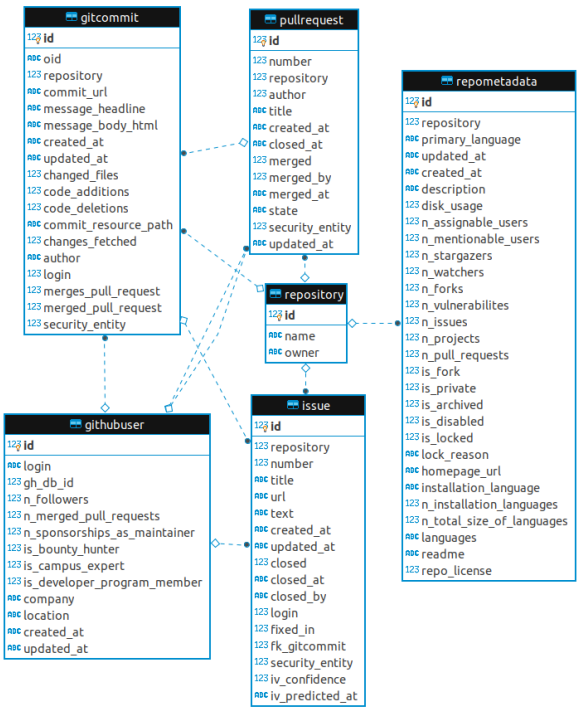


Figure A.1: Structure of the database where the raw GitHub data was stored.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-10-28

EXAMENSARBETE Exploring subjectivity in ad hoc assessment of open source software
STUDENT Jonathan Skogeby
HANDLEDARE Martin Höst (LTH)
EXAMINATOR Ulf Asklund (LTH)

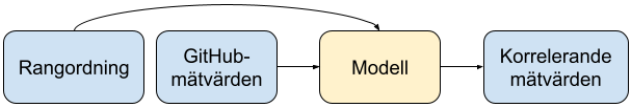
Representera subjektiva kvalitetsbedömningar av GitHub-projekt med en linjärviktad modell

POPULÄRVETENSKAPLIG SAMMANFATTNING **Jonathan Skogeby**

Kvalitetsbedömning av projekt med öppen källkod sker ofta på improviserat vis med hög nivå av subjektivitet beroende på vem som utvärderar och vilka omständigheter projektet utvärderas för. I detta arbete har en enkel modell utvecklats och testats med syftet att försöka identifiera de mätvärden som använts under kvalitetsbedömningen.

GitHub är i särklass dagens mest använda plattform för att lagra kod och samarbeta på mjukvaruprojekt med andra utvecklare. Många användbara mätvärden kan överskådas på GitHub-sidan för ett projekt: Antalet bokmärkningar, följare, utvecklare och rapporterade buggar, senast tillagt kod, vilken licens som används, med mera. Oftast använder sig utvecklare av dessa mätvärden för att fastställa kvaliteten hos projektet, och jämför sedan med andra alternativ för att dra slutsatsen om vilket projekt som ska användas för att uppnå önskad funktionalitet.

Att välja fel mjukvarukomponent kan bli kostsamt. Bristande säkerhet, nedläggning av projektet och buggar utgör några av de risker som kommer med mjukvara med öppen källkod. Därför vill man se till att välja rätt redan från början. Dagens mjukvara innehåller ofta flera hundra komponenter med öppen källkod och att manuellt utvärdera alla alternativ är inte hållbart. För att utforska hur processen kan automatiseras måste det först avgöras vilka delar av processen som går att uttrycka i kod. I detta examensarbete har en modell utvecklats för att utforska möjligheten att kartlägga vilka mätvärden på GitHub som använts av en viss utvecklare i sin kvalitetsbedömning. Baserat på data insamlad från GitHub och en utvecklares rangordning av en samling projekt kan den kombination av mätvärden som bäst korrelerar till rangordningen härledas av modellen. Följande figur ger en förenklad bild över modellens funktion.



Rangordningen av 20 projekt gjord av fem olika utvecklare samlades in tillsammans med mätvärdesdatan från samma projekt på GitHub. Resultatet pekade på att modellen ej lyckades fastställa vilka mätvärden som användes av utvecklarna när de rangordnade projekten trots att korrelationen mellan utvecklarnas och modellens rangordning var hög. Dock ger resultatet ej någon tydlig inblick i hur modellen hade presterat med en större datamängd.