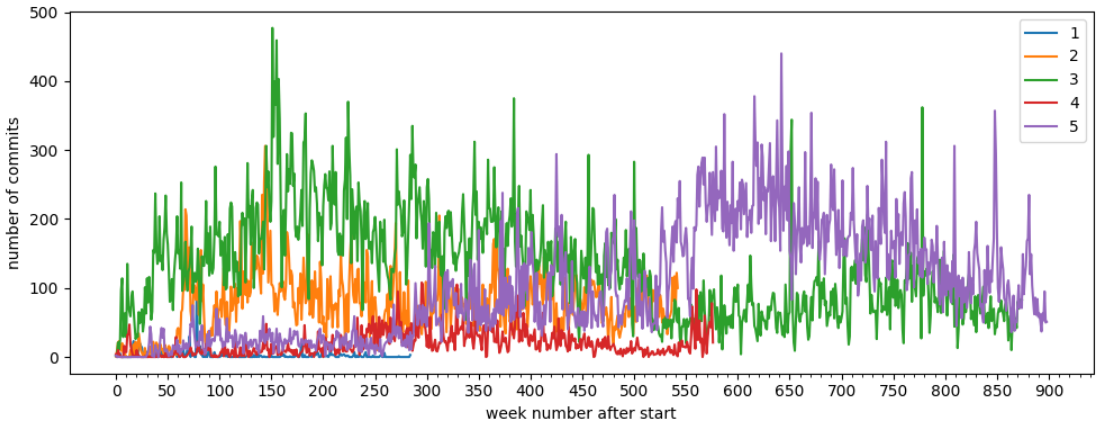

Clustering software projects at large-scale using time-series

A novel method that enables efficient large-scale analysis of software projects on commodity hardware

Master's thesis in Software Engineering and Technology

Heiko Joshua Jungen & Peter Pickerill

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Clustering software projects at large-scale using time-series

A novel method that enables efficient large-scale analysis of software projects on commodity hardware

Heiko Joshua Jungen & Peter Pickerill

Department of Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Clustering software projects at large-scale using time-series
A novel method that enables efficient large-scale analysis of software projects on commodity hardware
Heiko Joshua Jungen & Peter Pickerill

Supervisor: Miroslaw Staron, Department of Computer Science and Engineering
Examiner: Eric Knauss, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Clustering software projects at large-scale using time-series
A novel method that enables efficient large-scale analysis of software projects on commodity hardware
Heiko Joshua Jungen & Peter Pickerill
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Context Within the field of Mining Software Repositories, there are numerous methods employed to filter datasets in order to avoid analysing low-quality projects, such as backups and student projects. Since the rise of GitHub, the world's largest repository hosting site, large scale analysis has become more common. However, filtering methods have not kept up with this growth and researchers often rely on "quick and dirty" techniques to curate datasets.

Objective The objective of this thesis is to develop a method that clusters large quantities of software projects in a limited time frame. This explores the possibility that a fully-automated method can be used to identify high-quality repositories at large scale and in the absence of an established ground-truth. At the same time, the hardware requirements and time limitations of existing approaches should be reduced to remove the barrier for researchers.

Method This thesis follows the design science methodology. The proposed method, PHANTOM, extracts five measures from Git logs. Each measure is transformed into a time-series, which is represented as a feature vector for clustering using a k-means algorithm.

Results Using the ground-truth from a previous study, PHANTOM was shown to be competitive compared to supervised approaches while reducing the hardware requirements by two orders of magnitude. The ground-truth was rediscovered by several k-means models, with some models achieving up to 87% precision or 94% recall. The highest Mathews correlation coefficient (MCC) was 0.65. The method was later applied to over 1.77 million repositories obtained from GitHub and found that 38% of them are "well-engineered". The method also shows that cloning repositories is a viable alternative to the GitHub API and GHTorrent for collecting metadata.

Conclusions It is possible to use a fully automated, unsupervised approach to identify projects of high-quality. PHANTOM's reference implementation, called COYOTE, downloaded the metadata of 1,786,601 GitHub repositories in 21.5 days, which is over 33% faster than a similar study using a computer cluster. PHANTOM is flexible and can be improved further, but it already shows excellent results. The method is able to filter repositories very accurately with low hardware requirements and was able to rediscover an established ground-truth. In future work, cluster analysis is needed to identify the characteristics that impact repository quality.

Keywords: Software quality, Unsupervised Clustering, GitHub, Git, MSR, Time-series, Feature-based, Filtering

Acknowledgements

There are a number of individuals without whom, this work would not have been possible; We would like to thank our supervisor for sharing his knowledge and support. Munaiah et al. for their paper "Curating GitHub for engineered software projects"[25], which was the inspiration and baseline for this thesis. We also express our gratitude to the IT staff at the Chalmers University of Technology, for providing the resources needed to run COYOTE. We thank the researchers and faculty members in the department of Software Engineering at Chalmers University of Technology for their advice, time and expertise. We wish to thank GitHub and their support staff, who have helped us with technical questions and to reach an agreement on large-scale downloading.

Heiko Joshua Jungen & Peter Pickerill, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

Imagine you were given a large collection of software projects. How would you identify only those projects that have been developed to a high standard? Software project analysis is traditionally performed on a small corpus of selected projects, as seen in industry-based case studies [11, 40, 41]. Source code, metadata, and project artefacts have been used to judge quality or benchmark projects[20] against each other. In the recent years, due to the rise of GitHub, researchers have had access to a massive corpus of software projects that can be analysed. However, the quality of GitHub's over 80 million repositories is unclear[16]. It has been shown that most repositories can be considered to be of low quality, and could therefore skew analysis[23].

In the 2017 Mining Software Repositories (MSR) conference, a number of studies performed analysis on GitHub repositories. The reported dataset sizes ranged between one to over 80,000[7, 19, 26, 37, 48, 33, 24]. Most of these studies needed to filter out low quality repositories from the collection, where low-quality denotes those repositories that do not fit into the desired sample. Despite this, there is no standard approach to filtering in the MSR field. While filtering by popularity has been used a number of times, it has been shown to perform poorly[25]. It is clear then, that for researchers to make use of the large number of repositories available on GitHub, new filtering methods are required. In 2017, Munaiah et al. proposed a filtering method that outperformed traditional filtering approaches by using supervised classification. With this framework over 1.8 Million GitHub repositories have been analysed, one of the largest datasets to date[8]. While this is an impressive achievement, a number of key issues with the method remain. First, the required computing resources are out of reach of most researchers. Second, the required effort to establish a ground-truth was extensive. Third, some measures cannot be captured on every repository. Therefore, a new method is needed that can be used to filter repositories at large scale, using measures applicable to all repositories, and without requiring high computing resources and an established ground-truth. With this in mind, the authors propose one research question for this thesis; *Is it possible to cluster software repositories at large scale, based on their development history, while using commodity hardware?*

The authors argue that it is possible to achieve comparable results to Munaiah et al. by using an unsupervised clustering algorithm to filter repositories, therefore removing the need for researchers to establish a ground-truth. By simplifying the acquisition of information used for analysis, using the development history exclusively, it is possible to perform the analysis on commodity hardware. At the same time, the method shortens analysis time and achieves comparable accuracy to Munaiah

et al.'s approach. The method proposed in this thesis is named Project History Analysis of Time-Series Method, or PHANTOM for short. PHANTOM's efficacy is shown by using a ground-truth consisting of 650 labelled repositories and its applicability for large-scale analysis is shown when it is applied on a dataset of over 1.8 million software repositories. Both datasets and the ground-truth are published in [25], which is referred to as the baseline study.

The contribution of this thesis is; the PHANTOM method which compares repositories by using their development history. Second, the implementation tool COYOTE, which can be used to extract feature vectors and fit k-means models. Third, the Git logs, time-series, and feature vectors for over 1.77 million GitHub repositories.

The structure of this thesis is as follows; the background and related work are presented in chapter 2 and chapter 3. The problem statement and the design of this thesis are presented in chapter 4. The main chapter is chapter 5, which presents the requirements, design, datasets and validation of PHANTOM. The results of this are evaluated in chapter 6. The significance of the study, threats to validity, ethical considerations and future work are discussed in chapter 7. The thesis is concluded in chapter 8.

2
Background

In this chapter the background of the thesis is presented. The chapter is structured in sections that cover version control, time-series and machine learning, which are the main topics of this thesis.

2.1 Version Control

Version control systems (VCS) are tools that track file-level changes within software repositories. VCS allow multiple programmers to contribute to a repository at the same time. As VCS handle and store a history of these changes, they are a useful data source for researchers as they can trace the changes made to source code over time. VCS are a common component of software development and there are many variants (e.g. Git, SVN, Mercurial).

One example of VCS is Git. Git is a distributed version control system, that is open-source and free to use. It allows developers to manage a repository locally and synchronise with a Git server. Git introduces terminology that describes particular actions that developers perform on repositories[6]. A list with descriptions of the of the terms used in the thesis is shown in table 2.1.

One of the most common ways to use Git is through GitHub. Github is a repository hosting service that provides public and open-source repositories with a free online Git repository. As of March 2018, GitHub has over 24 million users and over 80 million repositories[16], making it the most popular code hosting website in the world[8]. In addition to hosting the repository, GitHub provides a number of additional details about the development of a repository (e.g. issues). It also provides a measure of the popularity of repositories through a feature called *starring*[18]. When a developer subscribes to a repository as a *stargazer*, they are showing their interest in the repository for others to see.

Due to Git's distributed nature, cloned repositories contain a record of every commit pushed to them. Git enables developers to see this record of these commits, and their authors, through the Git log. As GitHub is so popular, many projects use it to host a centralised repository for all push actions, rather than using the distributed features of Git. This makes GitHub an excellent source for repository metadata, and it has become a popular target for research[23] as it gives a detailed picture of the development history of a large corpus of open-source repositories.

GitHub repositories can be cloned by using the Git client. Passing the URL of a repository to the command *git clone* will download the repository onto the machine. To exemplify the structure of a GitHub URL consider `https://github.`

Term	Meaning
Commit	The action of submitting/contributing local changes into a specific branch.
Integration	The action of accepting a pull request into a specific branch.
Clone	Making a local copy of a repository that developers can work on.
Fork	Making a remote copy of a repository on GitHub. That is to say, a new repository on GitHub is created by copying an existing one.
Merge	The action of combining changes made to a file in different commits. Can also refer to the combination of two branches resulting in one branch containing all the changes of the second. This action may result in conflicts, which have to be resolved manually.
Pull	The action of retrieving commits from a remote repository.
Pull request	A request from one developer to another to merge commits into a repository.
Push	The action of updating a remote repository with changes made since the last pull.
Repository	The storage of the project files and historical records of those files.

Table 2.1: Terminology of the version control system Git.

`com/Netflix/SimianArmy`. This URL refers to the user *Netflix* who has created the repository *SimianArmy*. The convention for GitHub URLs is the GitHub domain followed by *user/repository*.

Git allows the formatting of logs by using the *git log --format* command. The format parameter takes a string with any combination of tokens (e.g. %ae for author email), which allows the user to choose which information to include within the outputted log. A complete list of tokens can be found at [15].

2.2 Time-series clustering

Due to the exponential increase in stored data, in part due to an increased ability to monitor processes over time, a popular way to represent and analyse sequential datasets is by using time-series. It is used in many fields (e.g. finance, economy, environment and transportation) to analyse data[28]. Time-series are sequential observations of a value, which are either made regularly or irregularly. Regular observations form an evenly spaced time-series, whereas irregular observations form unevenly spaced time-series. In fig. 2.1 an example of a regular time-series, consisting of fifteen observations, is plotted. As the time-series is evenly spaced it could be represented as a vector;

$$T = \{t_1, t_2, \ldots, t_{15}\}, \text{where } t_x \text{ are the values for the observations} \tag{2.1}$$

In table 2.2 the observations are listed to exemplify the data format of a time-series.

When clustering time-series there are three approaches[2]. The first is whole time-series clustering and is used in this thesis. In this approach, all observations

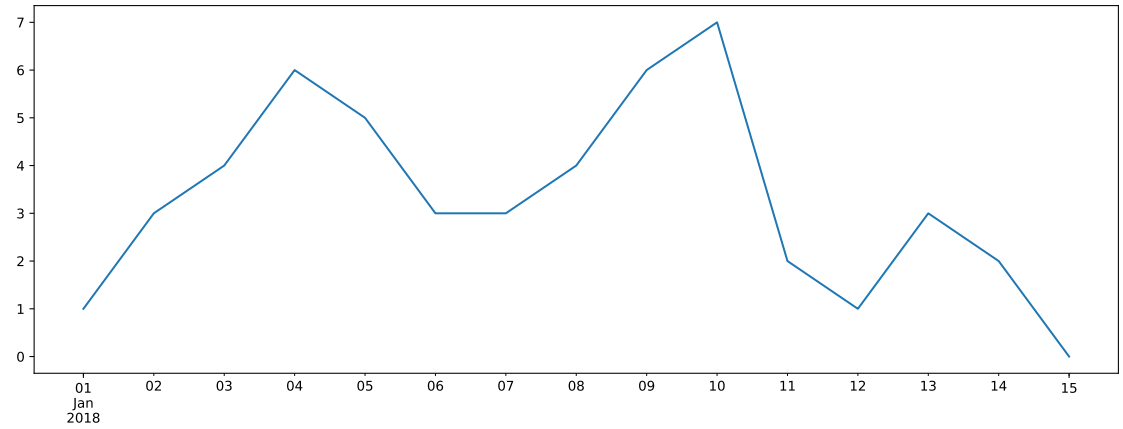

Figure 2.1: An example plot of a regular (evenly spaced) time-series. Values are shown in table 2.2.

Order	Date	Value	Order	Date	Value
1	18-01-01	1	9	18-01-09	6
2	18-01-02	3	10	18-01-10	7
3	18-01-03	4	11	18-01-11	2
4	18-01-04	6	12	18-01-12	1
5	18-01-05	5	13	18-01-13	3
6	18-01-06	3	14	18-01-14	2
7	18-01-07	3	15	18-01-15	0
8	18-01-08	4			

Table 2.2: An example table of a regular (evenly spaced) time-series.

are taken into account. The second approach is subsequence clustering which as the name suggests, compares subsequences of a time-series against each other. Finally, in the third approach, individual data points are compared. This approach is called time-point clustering. The approaches are summarised in table 2.3.

Due to the large variations in length, time-series are incompatible with traditional clustering algorithms, so one has to either customise clustering algorithms or convert time-series to different, simpler representations[2]. One way to do this is the feature-based approach. In this approach, a time-series is represented by a feature vector. This feature vector is usually fixed-length which makes it compatible with common clustering algorithms. A feature vector consists of a number of values; one example feature is the highest value in the time-series.

2.2.1 Euclidean Distance & Dynamic Time Warping

Euclidean distance, which is the distance of two points within Euclidean Space, can be used to compare time-series against each other. Each individual observation within a time-series can be compared to another observation in a different time-series via this measurement. However, Euclidean Distance has problems detecting

Clustering approach	Description
Whole time-series	Clustering of complete time-series.
Subsequences time-series	Clustering of data ranges from a time-series. Sequences can be compared within and across time-series.
Time-point clustering	Single observations are clustered either against points from the same or other time-series.

Table 2.3: The three approaches to time-series clustering.

patterns in time-series. Consider the time-series A and B;

$$A = 0, 1, 0, 0 \text{ and } B = 0, 0, 1, 0 \tag{2.2}$$

Calculating the Euclidean distance would not identify that the two time-series actually show the same trend, because they are slightly shifted [30].

A shift within a time-series would be recognised by the Dynamic Time Warping (DTW) algorithm, which is most commonly used when comparing time-series[30]. DTW first identifies an optimal many-to-many mapping between the observation of the time-series, which effectively stretches or compresses them (within certain bounds) so that they align better. In the case of A and B the mapping could look like this;

$$D(a_1, b_1), D(a_1, b_2), D(a_2, b_3), D(a_3, b_4), D(a_4, b_4) \tag{2.3}$$

where D is a distance measure (e.g. Levenshtein) between two observations.

When time-series have slightly different lengths, Euclidean Distance cannot compare the whole time-series. DTW can calculate the best mapping between data points. Data points may be mapped to one or more other points to compensate for the different lengths. However, when the time-series are of very different lengths, DTW may not be applicable as the boundary on how much the series can be stretched or compressed (warping size) would be very large. An alternative approach is to interpolate (i.e. insert estimated values) the shorter time-series to match the length of the longer one.

2.2.2 Extracting Features from time-series

Time-series can be characterised by features, which relate to particular attributes of the sequence. For example, a feature could be the lowest value within the sequence, which could be called *Min Y*. Features can show very simple, or very complex characteristics (see fig. 2.2). The up and down peaks of the time-series are marked using upward and downward pointing triangles. To calculate the features *Peak Up* and *Peak Down* these points are counted. A peak can be described as any point that is either higher or lower than the preceding and succeeding points. *Peak None* is the sum of all points that are not marked (e.g. at week 250). The feature *Max Y* is the largest value within the time-series, which is roughly 4000 in the example. The position of *Max Y* is captured by *Max Y Pos*, which is the index (week) in which the value occurred, around 150 in the example. *Duration* is equal to the total number

of weeks between the first point and the last point, illustrated by the bar close the x axis.

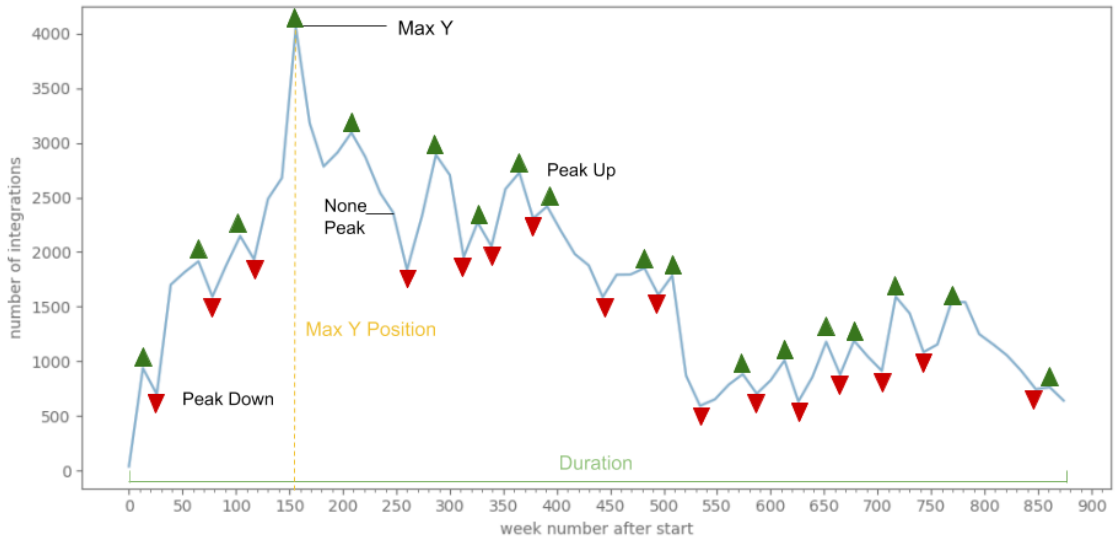

Figure 2.2: Example features that can be extracted from a time-series.

In fig. 2.3 more features are illustrated. At the start of the time-series a subsequence is labelled to show a positive gradient. All positive gradients between neighboring observations are averaged to calculate the feature *Mean Positive Gradient*. Similarly, the feature *Mean Negative Gradient* is calculated by averaging the negative gradients. A further set of features relate to amplitude; *Min Amp*, *Avg Amp* and *Max Amp*. Amplitude is the increase in value that is measured between an up peak and its previous point, divided by the *Max Y* value. That means, it is the increase relative to the maximum value. An example is shown at around week 250 in (b). The amplitude is labelled in the middle of the plot and for the purpose of the example, the difference between the peak and the previous value is equal to 1000. The *Max Y* value of the time-series is roughly 4000, which therefore means an amplitude of 25%. *Min Amp* is the lowest measured amplitude, *Avg Amp* is the mean of all amplitudes, and *Max Amp* is the highest amplitude.

2.3 Machine Learning: Classification and Clustering

In machine learning, statistical techniques are used to train an algorithm for a specific task. Although they are trained to a specific task, machine learning algorithms are general purpose and have been applied in many research areas and industries (e.g. medicine, business, computer science and social sciences)[36]. From a high-level perspective, machine learning algorithms start with a set of objects, learn from them and produce an output. Examples of input objects are images, text and sensor data. Outputs are dependent on the algorithmic family, where the three main ones

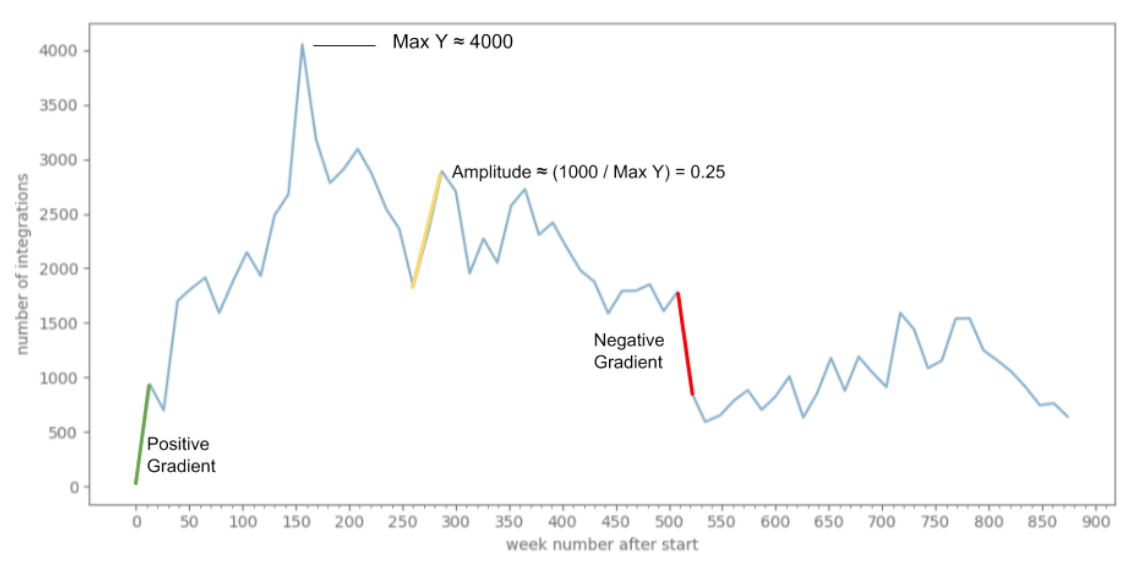

Figure 2.3: Example features that can be extracted from a time-series.

are; classification, clustering and projection [36]. Input objects are usually represented by a (fixed-length) feature vector that captures observation on the real-world object (e.g. pixels from an image, word counts from a text, values from a sensor).

2.3.1 Classification and Clustering

There are supervised and unsupervised machine learning algorithms. Supervised learning requires a ground-truth for the input data, which is used to learn the the structure of the input data. Unsupervised learning does not require a ground-truth and instead identifies the structures by itself. Classification belongs to supervised and clustering belongs to unsupervised learning.

The task of classification is to predict the class of an object. Class refers to a group or category that the object belongs to (e.g. an email is classified as either spam or not spam) and the class is indicated by a label[1]. In classification the starting point is a collection of objects where the class is known. These objects, together with their labels, become the input to a model that learns to correctly predict object labels. The objects are referred to as the training data and their labels are the ground-truth. After a model has learned the characteristics of the training data from the ground-truth, labels for new objects can be predicted.

In clustering, the input to the algorithm is a set of objects without labels. That means, in contrast to classification, there is no ground-truth associated with the input. The task of clustering is to group the input in a way that objects within a group (or cluster) are similar to each other, and dissimilar to objects in other groups[1]. One application of clustering is shopping recommendation systems, where items are suggested to a customer based on the shopping behaviour of other customers. In this scenario, the interest is not in the cluster label, but rather to identify which items similar customers buy. Similarly, when clustering repositories, researchers are often interested in repositories that are similar to each other rather than what the cluster represents, which can be determined by later analysis.

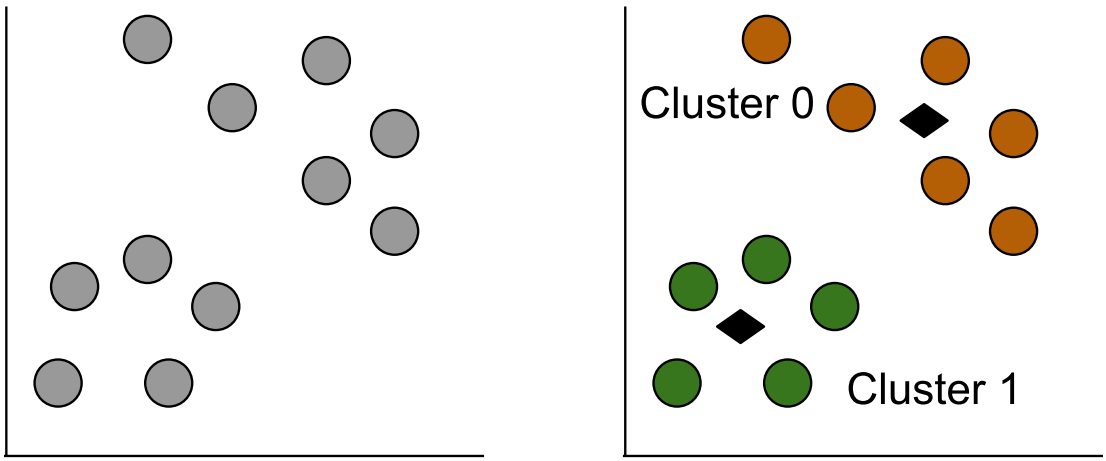

Figure 2.4: This figure illustrates the application of the k-means algorithm to a set of two-dimensional data.

Bandyopadhyay and Saha state that clustering consists of three steps; data acquisition, feature extraction, and clustering. Data acquisition is about gathering and pre-processing of data. In feature extraction, the dimensionality of the previously acquired data is reduced to a smaller set of "relevant features"[4]. A subprocess of the feature extraction step is to remove redundant information, and by that further reduce the number of features to a selected few. Finally, the selected features are grouped using a supervised or unsupervised algorithm[4].

2.3.2 K-means

Two commonly known clustering techniques are agglomerative hierarchical clustering and k-means[1]. However, Agglomerative hierarchical clustering does not work well at scale because of quadratic complexity (in the context of time-series clustering)[2].

The fig. 2.4 illustrates the k-means algorithm. On the left side, an example set of two-dimensional data is plotted. The right side shows the cluster centroids (indicated by the rhombuses), which exemplify the result of the k-means algorithm with the number of clusters set to two. The data points on the right side are labelled Cluster 0 and Cluster 1.

2.3.3 Accuracy

When predicting the label of an object and comparing the predicted label to the actual label (coming from a ground-truth) there are four cases, which are commonly known as error types; true positive(TP), false positive(FP), true negative(TN) and false negative(FN). The error types are used to calculate measures to indicate how well an algorithm performs the prediction; precision, recall and F-measure. Precision is the ratio of correctly captured data points (true positives) and all captured data points (true positives and false positives), by that describing how pure a cluster is;

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.4}$$

Considering the right illustration of fig. 2.4, and assuming that the data points belonging to Cluster 1 are truly belonging to this cluster (only true positives), the precision would be 100%. There are no data points which are falsely believed to be of this cluster (false positives). A high precision is desirable, because it shows that the algorithm accurately predicts object labels. Recall is a further measure that complements precision. It is the ratio of true positives, and the sum of true positives and false negatives;

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.5}$$

Recall is the percentage of data points that were captured by the cluster (true positives), compared to the total number of data points that should have been captured by the cluster (true positives and false negatives). This means, a high recall is desirable because it shows that many of the objects of a certain class are identified correctly. The F-measure is a compound measure, which is the average of precision and recall. F-measure is used to determine the overall effectiveness of the model.

$$\text{F-Measure} = \frac{\text{Precision} + \text{Recall}}{2} \tag{2.6}$$

Precision and Recall do not take all of the error types into account. An additional measurement, the Mathews correlation coefficient (MCC), provides a better picture of the classifier's performance when combined with the aforementioned measurements. It ranges from -1 to 1, where -1 indicates that the classifier is always wrong, 0 indicates it is random, and 1 indicates that it is always correct. The MCC is defined as;

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.7}$$

3

Related Work

Since 1997, the International Software Benchmarking Standards Group (ISBSG) has been collecting project-artefacts data from industry practitioners through questionnaires[20]. They have collected 8,200 projects so far, covering seven industry types[21]. Although the ISBSG is non-profit, they charge for data access, which combined with the small-scale could prevent researchers from using ISBSG data. It appears that a lack of automation limits large-scale analysis.

The implementations in [13] and [25] feature a high-level of automation, yet the resources required (e.g. time, computing power) are large. Gabel and Su conducted a study about the uniqueness of source code within C++, C# and Java applications taken from sourceforge[1]. They approached the question of *How unique is software?* by performing lexical analysis on a dataset of 6000 projects (in total 420 million lines of code). Following this, the percentage of unique code within 30 selected projects was measured. The study identified a general lack of uniqueness within software, where most programs are made from code snippets found in other software. The analysis time took four months, where source code was compared on a token-level with an optimised tool. This shows that source-code analysis requires a lot of time even for a small number of projects.

Metadata analysis is time-consuming at large-scale, primarily due to data collection limitations. Cosentino, Canovas Izquierdo, and Cabot looked into 80 studies that mined GitHub. They found that the two largest data sources, GHTorrent and GitHub API, were criticized by researchers. The GitHub API was said to be a source of problems, given request limitations and errors in the data returned. Cosentino, Canovas Izquierdo, and Cabot state that "the GitHub API request limit acts as a barrier to get data from GitHub"[8] which effects curated datasets (such as GHTorrent) and individual researchers. Many researchers criticise the size and up-to-dateness of services like GHTorrent. Cosentino, Canovas Izquierdo, and Cabot report that of studies they explored, only three looked at more than 100,000 repositories. These findings show that despite using these services, researchers struggle to collect up-to-date data at large scale.

Robles et al.[35] identified the similar issues with GitHub when collecting information about twelve million repositories. First, due to API limits, they calculated that it would have taken fourteen months to collect all of the data using a single API key, adding that "[...] this would have made the data gathering unfeasible". To gather the data in a feasible time, twenty keys were used. Secondly, 25% of the twelve million repositories had been moved or deleted between the time GHTorrent

[1] `http://www.sourceforge.com`

collected its data and the Robles et al.'s request to GitHub API, which wasted keys and analysis time. This shows that data collection time can be reduced by using volunteered keys for data collection, a practice employed by GitHub mirroring sites, like GHTorrent and Boa.

The study of Kalliamvakou et al. [23] presented thirteen perils of collecting, analysing and interpreting data from GitHub. The study shows that the assumption that every repository is a software project does not hold. In a random sample, just 63.4% of the repositories were related to software development. Furthermore, most repositories have low or zero activity. The study showed that only 25% of repositories were active for over 100 days. Kalliamvakou et al. note that even when repositories are active, it is not guaranteed that they use GitHub exclusively. This prevents the full development effort from being captured. When used for research, the API's hourly request limit, the risk of changes and the reduced amount of information create challenges. An additional problem is that the information from GitHub may not be reliable in every case. Kalliamvakou et al. explain that there are different ways to merge commits and GitHub cannot detect all of them. Therefore, some merges are not reported through the API. A further peril when using the GitHub API is, that unlike cloning with Git, the GitHub API does not redirect requests when a repository was moved. Accessing a moved repository with the API will result with a not found status code.

Nuñez-Varela et al. conducted a systematic review of 226 papers studying source code metrics. They identified that most studies considered one programming language and paradigm. Over 85% of the studies use object-oriented metrics. This is reflected in the available public datasets and the metric extraction tools. While the paper does not reason why researchers focus on one language, it indicates that cross-language analysis of source code may not be straightforward. Although, the use of metrics "theoretically can be applied to any language"[27], in practice it is complex and tools do not support all languages.

In 2017, Munaiah et al.[25] presented an evaluation framework that classifies GitHub repositories based on seven measures (dimensions). These dimensions cover metadata and source code and are used to to label repositories as "well-engineered" or not. In order to establish a ground-truth, Munaiah et al. manually labelled 650 repositories, of which 200 were used for validation. Once two classifiers were validated, the most accurate was selected. It took over a month to label 1.8 million software repositories. This is an impressive achievement and also makes the study one of the largest in the MSR field. Munaiah et al. concludes that according to this classifier, 24.07% of the analysed repositories contained a well-engineered software repository.

Aghabozorgi, Seyed Shirkhorshidi, and Ying Wah presented a decade review on time-series clustering in [2]. Common fields of where time-series clustering is used are change recognition, prediction or recommendation, and pattern discovery. There are a variety of approaches, representation methods, similarity measures, clustering algorithms, and evaluation measures presented in the paper.

Dynamic Time Warping(DTW) was studied by Ratanamahatana and Keogh[32], who identified a number of myths associated with the technique. One finding was that DTW is not slow, as is commonly mentioned, and the authors show that it

can achieve $O(n)$ complexity. Another finding was that interpolation is also shown to have little effect on the resulting accuracy of similarity searches that use DTW, however they note that the warping limit has significant effects on the accuracy.

In [45] a method for time-series clustering on global characteristics is proposed. Wang, Smith, and Hyndman extracted 9 features (based on statistical descriptions), from time-series to capture the global characteristics. Then, by using a greedy forward search algorithm, the best subset of features is selected to cluster on. The evaluation of the approach on benchmark datasets has shown that meaningful clusters were produced by this method.

Deng et al. have proposed a method for time-series classification based on three statistical features (mean, standard deviation and slope). These features are repetitively taken on subsequences of the time-series and the merged to build a feature vector. Using greedy forward search the best subset of features is selected. With this subset, the time-series of a benchmark dataset can be classified with high accuracy, and the method could outperform DTW as well[9].

A similar, but much more sophisticated approach to time-series classification is presented in [12]. In this paper, over 1000 features of time-series are extracted. Before classification, the best subset of features is determined by using a greedy forward algorithm. These features are then used as input into a classifier. This approach outperforms DTW "despite dramatic dimensionality reduction"[12] of the time-series. In some cases, one feature was sufficient to classify time-series with high accuracy.

In 2012, Esling and Agon looked into representations of time-series data. They state that "Defining algorithms that work directly on the raw time series would therefore be computationally too expensive"[10], which shows the need for simple time-series representation formats. The paper outlined 5 requirements for any time-series representation, such as low computational cost and good reconstruction quality. Although, they admit that representation methods make trade-offs between these requirements. Additionally, Esling and Agon highlight a need for time-series systems that do not require expert knowledge.

Guo studied trends in stock market time-series data. They clustered patterns as identified in the pricing of 30 stocks over 200 days. Using time-series and k-means, they were able to identify 4 distinct patterns despite high levels of noise and dimensionality[22].

4

Methodology

When filtering repositories there are a number of problems that can occur that PHANTOM needs to address. These problems are investigated in this chapter with a summary at the end. In addition, the research strategy for this thesis is presented.

4.1 Problem Investigation

An initial problem is deciding on what measures should be extracted from repositories in order to filter them. In MSR research, a number of measures have been suggested. Often, these measures are singular and do not take chronology into account. For example, number of stargazers is a single measurement approach, which has been shown to be ineffective at filtering for quality repositories[25]. Effective filtering methods strengthen conclusions because samples remain free of undesirable repositories. Therefore, it is important to address the problem of how to select measures that can be used to identify desirable repositories, such as those of high-quality.

Community based filtering strategies, for example; popularity, issues, and forks are common, but do not measure the internal quality of a project. These methods cannot be used on repositories without community interaction, for instance those that are hosted on private servers. Such repositories would be missing or misclassified in any analysis using these filtering methods. Therefore, it is important to address the problem of finding a uniform way of grouping similar repositories together, that does not rely on community engagement.

Most filtering methods trade depth for speed and simplicity. These methods are easier to implement, relying on a small number of measures without looking into the development history (appendix A). In [25], the in-depth methods were shown to require significant manual effort by multiple researchers to identify well-engineered repositories. With the exception of the proposed filtering method in [25], researchers need to choose "quick and dirty" filtering techniques, which have been shown to be inaccurate.

One clear problem is that computing resources are limited for most researchers, forcing them to choose methods that are not in-depth. *Reaper* required over a month to finish analysis on a computer cluster with over 200 nodes[25]. Therefore, new methods should reduce hardware requirements when conducting in-depth analysis compared to existing approaches.

Chronologic measures taken from repositories can be represented as a time-series. In the field of time-series clustering a number of techniques are available. Euclidean

Distance and DTW are most commonly used, however they cannot compare time-series of very different lengths. Feature-based approaches have been shown as an effective way to compare time-series of different lengths, and reduce the dimensionality of the data significantly. However, extracted features must describe the time-series correctly to ensure accurate clustering. Each method comes with its own set of limitations, which makes it difficult when choosing between them.

Researchers must decide between classification and clustering algorithms when grouping repositories. Classification requires an established ground-truth, which can be time-intensive to discover[25]. This ground-truth must be accurate to produce meaningful predictions. Clustering does not require a ground-truth, but requires later investigation. This investigation is needed to discover whether clusters are meaningful, which can also be time-intensive.

Based on this, the authors have identified five main problems that the new method should address;

1. Which measures should be taken from repositories
2. How to transform time-series to feature vectors
3. Quality of the established ground-truth for software repositories is unclear
4. How to cluster feature vectors using unsupervised learning algorithms
5. Current methods perform poorly on commodity hardware

4.2 Research Methodology

The research conducted in this thesis follows the design science methodology. In design science, research follows the engineering cycle. It is called a cycle because the research is conducted iteratively, which means the steps are performed multiple times until the objectives are reached. The engineering cycle is a "rational-problem solving process"[46] and consists of five steps:

1. Problem Investigation
2. Treatment Design
3. Treatment Validation
4. Treatment implementation
5. Implementation evaluation

The design cycle, which is a sub-cycle of the engineering cycle, is described in this paragraph as summarised from Wieringa[46] and the explanations are are not further cited for brevity. The cycle begins with an investigation of the problem, to gain an in-depth understanding of the causes. The acquired knowledge is then used to design a treatment. Treatment is defined as the interaction of an artefact with a problem context. For this thesis, the artefact is PHANTOM, the proposed method

to analyse software projects, and the problem context is to filter well-engineered repositories from GitHub. Part of the treatment design is to specify requirements for the treatment. The goal of the third step, treatment validation, is to confirm that the designed treatment satisfies all the requirements and whether the treatment is able to treat the problem (i.e. filter repositories accurately). After treatment validation, the design cycle is finished and can be repeated if needed. Leaving the design cycle and continuing to the fourth step, treatment implementation, the designed treatment is applied to the problem context. For this thesis, this means that a reference implementation of the proposed method is implemented to cluster repositories from GitHub. The final step of the engineering cycle is to evaluate the results produced in the preceding step. According to Wieringa, design science is restricted to design cycle (step 1, 2 and 3)[46]. However, this thesis surpasses the design science methodology, by performing the the full engineering cycle.

5
PHANTOM

The objective of this thesis is to develop a new filtering method that can address the problems identified in section 4.1. This method will use a time-series representation to capture historical information on projects through their repositories. The authors call this method the Project History Analysis of Time-Series Method (PHANTOM). In this chapter, the requirements for PHANTOM are stated and its design decisions are outlined. Then, PHANTOM is validated against the requirements using a reference implementation of the method, called COYOTE.

5.1 Requirements

PHANTOM will not have access to additional information beyond the Git repository itself, therefore any measures that are taken cannot rely on additional information (such as the GitHub API) (**R#1**). Time-series cannot be clustered directly due to variation in duration, therefore feature vectors are taken. However, these features must characterise the time-series well for accurate clustering (**R#2**). PHANTOM should not require a ground-truth in order to be useful for researchers, but the produced clusters have to be meaningful. Therefore, PHANTOM should rediscover an established ground-truth to show the meaningfulness of the clusters (**R#3**). PHANTOM uses a standard k-means algorithm to cluster repositories, but without a ground-truth the accuracy of predicting new repositories is unknown. Therefore, k-means must be compared against a supervised algorithm which has been trained on an established ground-truth, to see if the unsupervised approach is competitive(**R#4**). Finally, PHANTOM is considered an in-depth method due to its investigation of development history. However, other in-depth techniques have hardware requirements that hinder general use, especially at scale. Therefore PHANTOM must show that it is possible to be in-depth while having hardware requirements within reach of the average researcher (**R#5**).

Below are the five requirements for PHANTOM;

R#1 All Measures must be extractable from the Git Log.

R#2 Time-series must be characterised by feature vectors accurately.

R#3 Show that an established ground-truth can be discovered using an alternative technique.

R#4 Show that k-means can compete with supervised algorithms.

R#5 Show that an in-depth method can perform well on commodity hardware at large-scale.

5.2 Design decision

The design decisions for PHANTOM are presented in this section. The complete process is illustrated in fig. 5.1 and the steps are explained in the subsequent paragraphs. The input to PHANTOM is a collection of repository URLs, which locate the repositories to be analysed. Each repository is cloned to the machine using Git (Step A). Next, the Git log of the cloned repository is generated (Step B) in the format defined in table 5.1, where each column is separated by a comma (","). From now on, Git log will refer to this format.

An example Git log is presented in table 5.2. The Git log contains timestamped rows, which makes the conversion to time-series possible (Step C). These time-series use combinations of the different columns of the Git log and are referred to as measures. In table 5.3 these measures are explained and the information used to obtain them is listed.

In table 5.4 the example Git log is transformed into the five measures. Measures are represented as a regular time-series, which makes their comparison possible; however, considering very different lengths, a direct comparison of the time-series may not make much sense. Interpolation would mean that the data are manipulated, which the authors argue would not be a true representation of the development history. The time-series are of very different lengths (e.g. 50 weeks and 900 weeks). Therefore, DTW and Euclidean distance are not suitable. Instead, a feature-based approach is selected, which does not come with these issues.

The time-series are therefore reduced in dimensionality by extracting a fixed-length feature vector (Step D). A complete list of the 42 extracted features is presented in in appendix B. The measures are extracted separately, which means that there is one feature vector per measure. In table 5.5 a sample of the extracted features is presented. Each repository in the input collection is processed in this way. After this, feature vectors are used in the subsequent steps.

Some features cannot be extracted from all measures. For example, peak-related features cannot be measured on time-series with a length of three or less, because peaks cannot be detected. Where features are immeasurable the value is set to zero ("0") during the preprocessing step (Step E), because the k-means algorithm cannot handle missing values.

The next step is to select the best subset of features from the feature vector

	Hashes		Author			Committer		
Name	Commit	Parent	Name	Email	Date	Name	Email	Date
Format	%H	%P	%an	%ae	%at	%cn	%ce	%ct
Type	GUID	GUID	String	String	UTS	String	String	UTS

Table 5.1: The format of the generated Git logs, each column is separated with a comma. UTS refers to Unix Timestamp, GUID refers to Globally Unique Identifier.

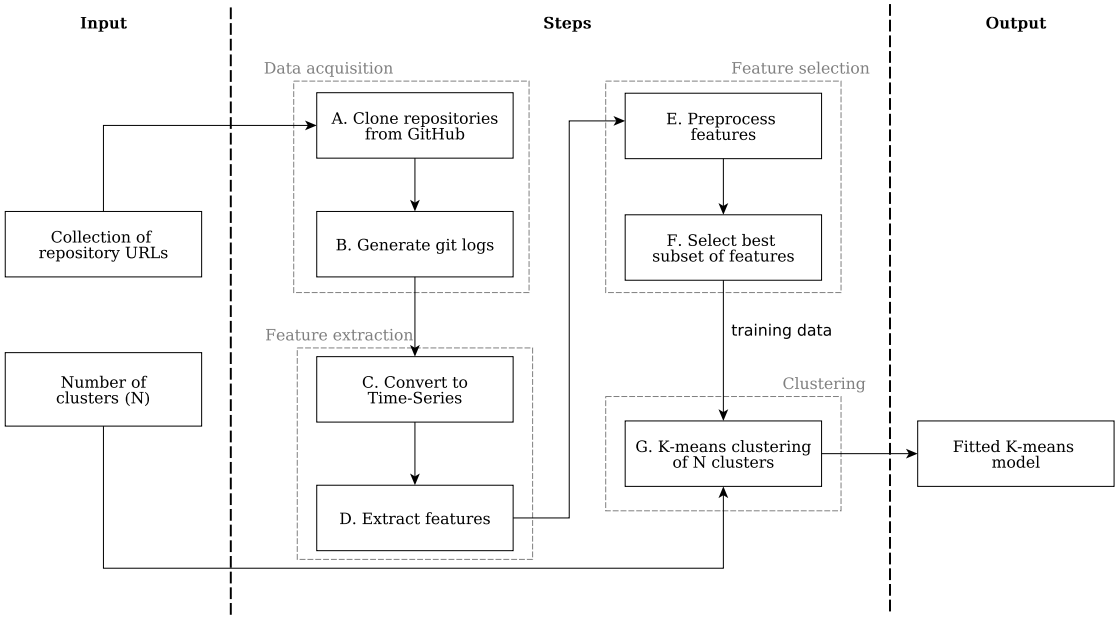

Figure 5.1: Overview of PHANTOM.

Hashes		Author			Committer		
Commit	Parent	Name	Email	Date	Name	Email	Date
b57f4f3	82c9f95	ab	a@b.com	1519904296	cd	c@d.com	1519904396
82c9f95	efaf9cd	ab	a@b.com	1519834072	ab	a@b.com	1519904296
efaf9cd	703b7b1	ab	a@b.com	1519404672	ab	a@b.com	1519824672

Table 5.2: Example Git log, where each row is one commit.

Measure	Git Log Information Used	Description
Commit Frequency (Commits)	Author Date	The number of commits summed per Week
Integration Frequency (Integrations)	Committer Date	The number of integrations summed per Week.
Commit Frequency (Committers)	Author Date, Author Email	The number of unique developers (by email) that have made commits per Week
Integration Frequency (Integrators)	Integrations Date, Integrations Email	The number of unique developers (by email) that have made integrations per Week
Merge Frequency (Merges)	Parent Hashes, Committer Date	The number of merges summed per Week

Table 5.3: The measures extracted by PHANTOM.

Date	Integrations	Integrators	Commits	Authors	Merges
2018-02-26	2	2	2	1	0
2018-02-19	1	1	1	1	0

Table 5.4: Example time-series for the five measures.

Feature	Duration	Avg Y	Max Y
Integrations	2	1.5	2
Integrators	2	1.5	2
Commits	2	1.5	2
Authors	2	1	1
Merges	2	0	0

Table 5.5: Example feature vectors for the five measures.

(Step F). In order to remove redundant features, the pearson correlation coefficient is calculated. If the correlation meets or exceeds a specified threshold the feature is removed.

The remaining features are normalised with the standard score and then used to fit a k-means model (Step G). The configuration parameters of the k-means model are presented in appendix D. Finally, the fitted model is outputted.

5.3 Datasets

Munaiah et al. published five datasets as part of their research "Curating GitHub for engineered software projects". These datasets are formatted as collections of GitHub repository URLs. Four of these datasets (organisation, utility, negative instances and validation) are used as ground-truths in this thesis, with the fifth being referred to as the large dataset (a collection of over 1.85 million URLs). To create the ground-truth datasets Munaiah et al. followed a manual curation process in order to label repositories. Each repository was independently judged by two or three researchers as either well-engineered or not, according to agreed guidelines. If the judgement about a repository differed, it was either discussed further or discarded. The datasets are summarised in the list below;

- Organisation

  The organisation dataset consists of 150 well-engineered repositories. Well-engineered repositories are defined as similar to those of popular software engineering companies such as Amazon, Apache and Facebook. The researchers manually investigated repositories to find those that matched the definition.

- Utility

  The utility dataset contains 150 well-engineered repositories. It defines a well-engineered repository as one with a general-purpose. That is to say, a reposi-

tory that has value to users other than the developers. The repositories were randomly sampled from 1,857,423 GitHub repositories.

- Negative Instances

  The dataset of negative instances holds 150 repositories that are not well-engineered. The repositories do not conform to either of the definitions of well-engineered. The dataset resulted from the selection process of the utility dataset, which means that it contains the first 150 repositories that both authors rejected.

- Validation

  The validation dataset consists of 100 well-engineered and 100 not well-engineered repositories. The selection process is similar to the one of the utility dataset and shares the definition of what is well-engineered and not.

- Large dataset

  This dataset is a collection of 1,857,423 GitHub URLs. In contrast to the other datasets there is no ground-truth, meaning the quality of the repositories is unknown.

5.4 Validation

This section validates the requirements stated in section 5.1. Each of the five requirements is discussed separately in the following subsections. **R#1** is validated against the design decisions, the remaining requirements are validated against the results of a reference implementation of PHANTOM. This implementation is referred to as COYOTE, which stands for Classification of Time-Series.

5.4.1 All Measures must be extractable from the Git Log

The five measures extracted by PHANTOM are; Integration Frequency, Commit Frequency, Integrator Frequency, Committer Frequency, and Merge Frequency. Each measure uses different parts of the Git log, along with at least one of the two date types (author or committer date). The other parts of the Git log are the author and committer email, and the number of parent commits (table 5.3). All of this information is available in every Git managed repository, and in fact Git ensures its availability, because when committing changes, the information is automatically recorded. PHANTOM has no dependency on additional data from GitHub, such as the GitHub API and mirroring services like GHTorrent. Due to the decision to use this specific set of information, PHANTOM is able to extract all measures from Git logs exclusively.

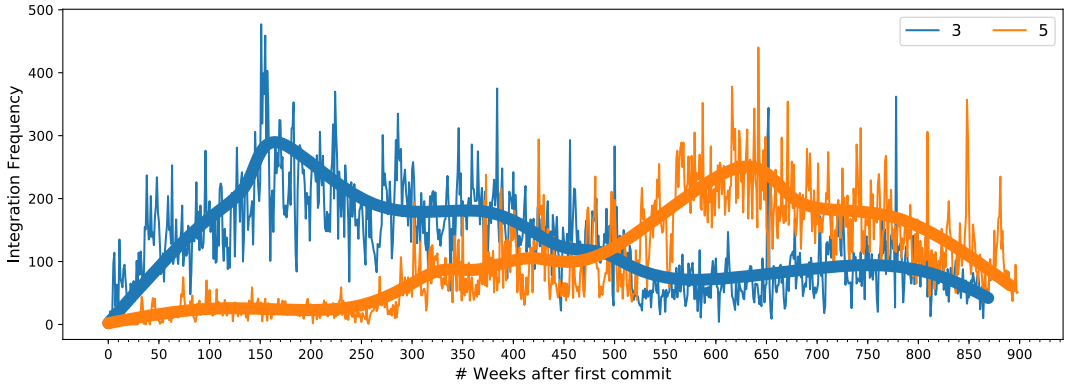

Figure 5.2: Integration Frequencies for `https://github.com/mono/mono` (3) and `https://github.com/FFmpeg/FFmpeg` (5).

Repository	Duration	Max Y	Max Y Pos
3	870	470	160
5	900	430	640

Table 5.6: Example feature vector for the time-series in fig. 5.2. The number have been rounded for illustration.

5.4.2 Time-Series must be characterised by Feature Vectors accurately

Feature vectors must capture the characteristics of time-series accurately to avoid k-means mislabeling them. It can be difficult to select features that do this. This problem is illustrated by the two integration frequencies plotted in fig. 5.2, which were selected from an investigation of twenty repositories using COYOTE. The plots are visually distinguishable. However, when converted into feature vectors, a small number of features, such as *Max Y* or *Duration* may not be enough to differentiate two time-series from each other (see table 5.6). The difference between the *Max Y* values and the *Duration* values is 30 and 40 respectively. *Max Y* and *Duration* are close enough that one can say that the time-series are similar to each other. Therefore, crucial features are missing to differentiate them. An additional feature such as the x value of the highest peak (*Max Y Pos*) would show a clear difference between the two repositories. PHANTOM uses *Duration*, *Max Y*, and *Max Y Pos* along with 39 other features (appendix B) to characterise time-series.

Even if a number of features are similar, the chances of an identical feature vector for a different time-series is reduced with a larger feature vector. By this, the k-means algorithm is able to cluster time-series via feature vectors effectively, as those differences are clear. By using larger feature vectors, PHANTOM captures the characteristics of time-series and the differences between them are highlighted.

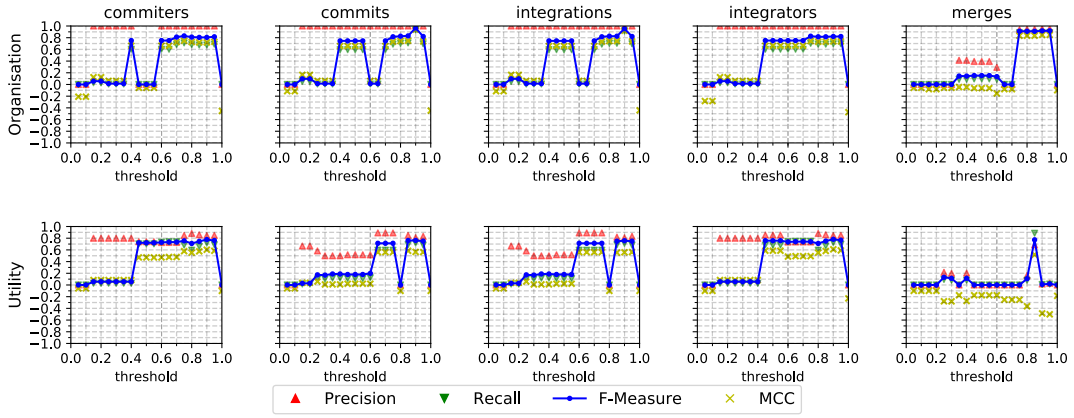

Figure 5.3: The accuracy of k-means model thresholds against the the ground-truth for the five measures.

5.4.3 Show that an established Ground-Truth can be discovered using an alternative Technique

In this section, COYOTE is used to fit k-means models on the ground-truth datasets. These are the organisation and utility repositories, which are both complemented with the negative instances, so that the datasets contain well-engineered and not well-engineered repositories to almost equal parts. PHANTOM requires a correlation threshold to select the best subset of features. As it is unknown which threshold is best, COYOTE explores thresholds ranging from 0.05 to 1, with a step size of 0.05. This means that for each combination of datasets and measures, twenty models are fitted. As k-means is unsupervised, the true labels are not known to the algorithm when fitting the model, which enables a comparison of the produced cluster labels and the ground-truth labels.

This comparison results with the four error types which are used to calculate the precision, recall, and F-measure (see fig. 5.3). On the organisation dataset there are many models that achieve a precision and recall close to 100%. On the utility dataset, the accuracy is lower with precision and recall of up to 90%. The high precision and recall indicate that the models were able to rediscover the the majority of true labels for both datasets. Overall, the organisation repositories could be rediscovered with higher accuracy than utility repositories. However, the accuracy largely depends on the dataset, measure and correlation threshold. This shows that a ground-truth could successfully be discovered using an alternative, unsupervised technique.

5.4.4 Show that K-means can compete with supervised Algorithms

In the baseline study a Score-based and Random Forest classifier was trained on the organisation and utility ground-truth datasets. The classifiers were then used to predict the validation dataset. In order to compare k-means to these algorithms,

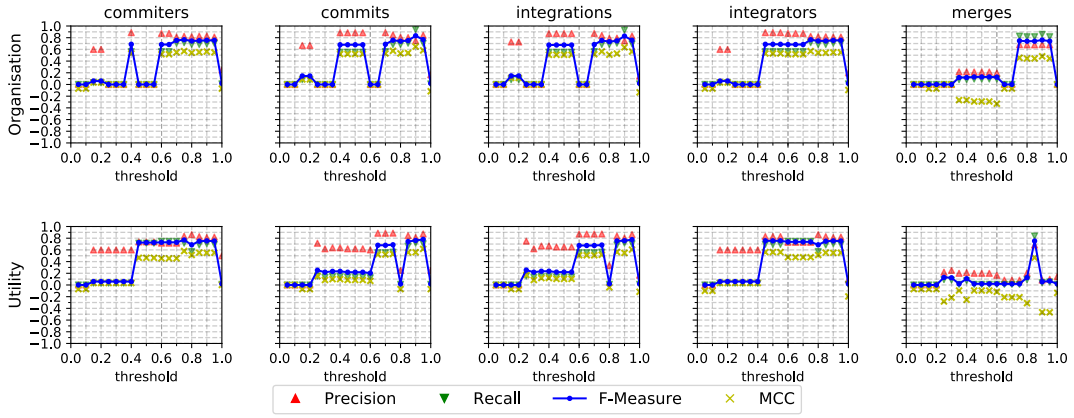

Figure 5.4: The accuracy of k-means model thresholds against the the validation data for the five measures.

Measure	Threshold	Precision	Recall	F-Measure	MCC	Features
Merges	0.90	0.68	0.85	0.76	0.48	17
Integrators	0.75	0.82	0.71	0.77	0.57	9
Integrations	0.90	0.75	0.93	0.83	0.64	19
Commits	0.90	0.75	0.94	0.83	0.65	19
Committers	0.75	0.82	0.71	0.77	0.57	11

Table 5.7: Prediction accuracy for the five measures when clustering repositories. (Organisation models)

COYOTE is applied to the same datasets. First, similarly to section 5.4.3, COYOTE explores a range of thresholds for each combination of datasets (organisation, utility) and measures. This time however, the fitted models are used to predict repositories from the validation dataset. Accuracy when predicting repositories is shown in fig. 5.4. As already seen in fig. 5.3, the accuracy varies across measures, datasets and thresholds.

To compare the results against the baseline study, the best models for each dataset and measure are selected. In order to achieve this the authors established a set of rules that determine the best model;

1. Find the highest F-measure
2. Find the highest precision
3. Find the highest recall
4. Find the lowest threshold

These rules are implemented in COYOTE which automates the feature selection process. In table 5.7 and table 5.8 the best models (fitted to the organisation and utility repositories respectively) are presented.

In comparison to the classifiers of the baseline study (table 5.9), it is clear that the COYOTE models can compete with the supervised approaches. The baseline

Measure	Threshold	Precision	Recall	F-Measure	MCC	Features
Merges	0.85	0.68	0.84	0.75	0.47	11
Integrators	0.45	0.83	0.69	0.76	0.56	5
Integrations	0.95	0.86	0.70	0.78	0.61	22
Commits	0.95	0.87	0.70	0.78	0.62	22
Committers	0.75	0.83	0.71	0.77	0.58	8

Table 5.8: Prediction accuracy for the five measures when clustering repositories. (Utility models)

Data set	Classifier	Precision	Recall	F-Measure
Organisation	Score-based	0.76	0.61	0.68
Organisation	Random Forest	0.88	0.42	0.57
Utility	Score-based	0.58	0.99	0.73
Utility	Random Forest	0.82	0.86	0.84

Table 5.9: The accuracy of the baseline's tool *reaper* as reported in [25].

classifiers, trained on the organisation dataset, achieve lower F-measure than any COYOTE model fitted to the same data. The models match the precision and surpass the recall of the supervised algorithms. Classifiers trained on the utility dataset set a higher benchmark than classifiers trained on the organisation dataset. COYOTE matches the F-measure of the Score-based classifier. Although the highest precision on the utility dataset of 82% could be surpassed by two out of five COYOTE models (Integrations, Commits), the recall on these cannot compete with the Random Forest (RF) classifier. Considering the precision and recall of the Committers model, it is clear that the accuracy is close to the accuracy of the RF classifier. This shows, that k-means is a competitive alternative to supervised algorithms.

5.4.5 Show that an in-depth Method can perform well on commodity Hardware at Large-Scale

COYOTE was used to download the organisation, utility, negative instances and validation datasets. In order to validate that PHANTOM is applicable at scale, the timings of COYOTE are extrapolated to the size of the large dataset. Since the organisation dataset comes from a different selection process than the others (see section 5.3) it is not used for this extrapolation.

COYOTE's results (table 5.10) show that 3.8% of the repositories were unavailable, which means they have been deleted or made private. The total download time was 7.5 minutes for 500 repositories, or an average of one repository every 0.94 seconds.

When these values are extrapolated up to the large dataset (1,857,423 repositories), the time to obtain the Git logs is estimated to be 20.2 days. This extrapolation validates that PHANTOM can perform well on commodity hardware, even at large scale.

Dataset	Available Repositories	Time Taken (minutes)
Organisation	149 / 150	10:39
Utility	145 / 150	2:44
Negative Instances	138 / 150	1:53
Validation(Well-engineered)	100 / 100	1:36
Validation(Not well-engineered)	98 / 100	1:18

Table 5.10: Number of available repositories and timings to clone and generate the Git log for them.

6
Results

In this chapter, the results for COYOTE's application on the large dataset are presented. COYOTE took 21.5 days to obtain the Git logs for 1,780,773 (95.36% of all) repositories. This leaves 76,650 (4.64% of all) repositories that were not available, due to either deletion or being made private. When converting the Git logs to time-series, some of the logs had to be excluded from the analysis, because of a formatting problem; Logs with author and committer names that contain a comma (",") had to be excluded, because the additional comma made the correct separation of information impossible since Git logs are saved as CSV files. For this reason 9,606 (0.5% of the obtained) Git logs had to be discarded. The remaining 1,771,167 Git logs were converted to time-series and feature vectors were extracted.

In order to predict the labels for the repositories a fitted model is required. Therefore the models that performed best on the validation dataset are selected. This means, ten models (one for each combination of ground-truth dataset and measure) are used to predict the repository labels for the large dataset. The models are chosen following the same set of rules as presented in section 5.4; however, since k-means uses random initial centroids, the accuracy varies, from the one seen in the validation. In table 6.1 and table 6.2, the best models and the number of repositories predicted to be well-engineered are presented. Out of these models, six resulted in a percentage of well-engineered repositories between 35% and 40% and two resulted in 55%. The remaining two models resulted with 19% and 96%.

6.1 Evaluation

In this section, COYOTE's performance is compared to *reaper*, the baseline study's implementation. COYOTE produced ten models where six were within a 5% range

Measure	Threshold	Number of well-engineered	Percentage of well-engineered
Merges	0.9	343,818	0.19
Commits	0.9	704,995	0.4
Committers	0.75	688,260	0.39
Integrations	0.9	700,501	0.39
Integrators	0.75	688,260	0.38

Table 6.1: Results from COYOTE on 1,771,167 repositories from the large dataset. Each row represents one model (Organisation models).

Measure	Threshold	Number of well-engineered	Percentage of well-engineered
Merges	0.85	1,716,118	0.96
Commits	0.95	985,213	0.55
Committers	0.75	690,371	0.39
Integrations	0.95	983,553	0.55
Integrators	0.45	617,042	0.35

Table 6.2: Results from COYOTE on 1,771,167 repositories from the large dataset. Each row represents one model (Utility models).

of one another. These six models predicted 38.33% (mean) of repositories to be well-engineered. This is significantly higher than the prediction for the baseline study, which was 24.07%.

COYOTE achieved a 33% reduction in data collection time over reaper, however 4.64% of repositories were unavailable. It took 21.5 days to generate the Git logs for the large dataset, or one second per repository, which is within 1.3 days of the extrapolated analysis time of 20.2 days (see section 5.4.5). At one second per repository, it would take around 2.54 years to clone all of GitHub. However, considering that GitHub will grow in this time, even at the this rate it would be infeasible to analyse all of GitHub. It is also interesting to note that for every extra second the analysis takes, a total of 21.5 days is added to the total analysis time for the large dataset. Therefore, it is important to reduce the analysis time for each repository when operating at scale. The data collection time concerns Git log retrieval only.

COYOTE reduced the hardware requirements over *reaper* by two orders of magnitude. *Reaper* analysed the large dataset using a computer cluster of 200 nodes, while COYOTE achieved the same using a standard desktop computerappendix C. Furthermore, the authors found that the hardware resources were not exhausted (RAM, CPU), spending most of the time idle. The majority of "analysis" time is spent waiting for downloads to complete, rather than Git log extraction.

Bandwidth is not a limiting factor in analysis, rather the download speed provided by GitHub is. The bandwidth available to the machine on which COYOTE ran was 1 Gbps. The authors observed the download speed, which rarely exceeded 40Mbps. This shows that bandwidth is not the constraint one might expect it to be, but rather the speed at which repositories can be downloaded from GitHub is.

COYOTE reduces the number of measures needed for analysis from seven taken by reaper to one. Although five measurements were experimented with, four show competitive results when used on their own. This shows that with only limited information, accurate predictions can be made about the quality of a repository. For instance, COYOTE can produce competitive results with Commit Frequency alone, which is also taken by *reaper* as part of the seven extracted measures (as a monthly average). This shows that COYOTE was able to achieve similar accuracy, with a subset of the data used by reaper (i.e. one seventh), by choosing a different representation.

Since measurements are taken from the Git logs, rather than other sources (e.g. source code, GitHub API, GHTorrent), private or closed-source repositories can now

be cross-analysed with open-source ones. PHANTOM also avoids the limitations of other sources such as out-of-date information and API key sharing.

PHANTOM's working assumption is that the programming language of a repository is not relevant. As the Git log is independent of the programming language, COYOTE can analyse any programming language, which is a significant improvement over reaper. However, the authors must admit that the efficacy of COYOTE on other programming languages has not been established, since the large dataset contains repositories from a set number of languages, which are the ones supported by reaper.

The above presented points are presented in table 6.3 and compared side-by-side to reaper.

Aspect	*Reaper*	PHANTOM
Data Collection Time	>1 month	3 weeks
Hardware Requirements	Computer cluster (200 nodes)	Desktop computer
Measures	7 measures	1 measure
Data Sources	GHTorrent, source code	Git
Sample Size	1,857,423	1,771,167
Machine Learning approach	Supervised	Unsupervised
F-Measure (Organisation)	68%	77%
F-Measure (Utility)	84%	76%
Percentage well-engineered	24.07%	38.33%
Programming Languages Supported	C, C#, C++, Java, PHP, Python, Ruby	Any
Implementation Languages	Python	Python, Rust

Table 6.3: Comparison between *reaper* and PHANTOM.

7
Discussion

In this chapter the significance of the study, threats to validity, ethical considerations and future work are discussed.

7.1 Significance of the Study

The amount of available software repositories rises everyday; between January 2014 and March 2018 the number of GitHub repositories rose from 10.6 million to over 80 million, an increase of 780%. This dramatic increase has not been matched by analysis methods so far. In order to make use of this large corpus of data, it is essential to filter out undesirable repositories, however as Munaiah et al. puts it: "there are limited means of separating the signal (e.g. repositories containing engineered software projects) from the noise (e.g. repositories containing homework assignments)"[25].

PHANTOM addresses this problem and improves on existing methods with respect to analysis time and hardware requirements, without sacrificing accuracy. Therefore, the barrier for researchers, who operate in short time frames and without expensive hardware, is removed. PHANTOM allows researchers to automatically and inexpensively curate desirable repositories for more specific analyses. In the MSR field, the authors observed the use of inaccurate or unproven filtering methods (e.g. popularity). PHANTOM could be applied in such studies to improve the data curation process. For example, Robles et al. published a collection of 24,000 repositories, for which PHANTOM could be useful to filter out undesirable repositories. Such use cases are possible, because PHANTOM is not dependent on mirroring services like GHTorrent. Any Git repository, not just those available through such services, can be analysed, making PHANTOM also suitable for research on private, or very specific collections of repositories. Furthermore, PHANTOM is programming language agnostic.

As part of this thesis several datasets are published; Git logs, time-series(measures), and feature vectors for both the ground-truth (630 repositories of known quality), and the large dataset (1,771,167 of unknown quality). These datasets can be used to reproduce the findings and carry out other analysis. For instance, they could be complemented with programming language information to find correlations between the quality and the programming language of software projects.

We hope that PHANTOM is used by other researchers as a baseline to improve large-scale analysis. This thesis has shown that an alternative way of analysing software repositories is feasible. However, the proposed method can be improved

and more algorithms experimented with.

7.2 Threats

External Validity The ground-truth [25], which is based on a description of 300 well-engineered and 150 not well-engineered repositories may not agree with other collections of repositories. Although the authors cannot confirm the correctness of the ground-truth, as it is not feasible, COYOTE does not use the ground-truth to train, because k-means is an unsupervised algorithm. The produced clusters agree with the ground-truth to a large degree, which supports its correctness. That being said, the possibility also exists that both COYOTE and the ground-truth are wrong and this agreement is coincidental. COYOTE was also not validated against other datasets, which could mean that it is overfitted, and therefore may not perform as accurately on other datasets. Furthermore, statements about the download speed may not be relevant to researchers with different hardware, internet connection, or an alternative agreement with GitHub.

Internal Validity The results of the thesis are based on the five measures that are taken from Git logs; Integration Frequency, Commit Frequency, Integrator Frequency, Committer Frequency, and Merge Frequency. These have been selected by the authors and may not reflect the true characteristics of repositories. In addition, feature vectors contain 42 features, which are also chosen by the researchers. Although attention has been paid to choose features that are reflective of the time series, no rigorous process was followed to ensure they were so.

7.3 Ethical Considerations

The most important ethical consideration concerns the Git logs published as part of this thesis, which contain the names and emails of GitHub users. Although these are publicly available, the authors have anonymised this data to protect the users' privacy, in accordance with GitHub terms and conditions. Therefore, the published Git logs contain placeholder names and emails which neither hinder analysis, nor leak sensitive information.

A further ethical consideration is the collection of repositories from GitHub. Although the repositories are publicly available, mining data on the scale seen in this thesis is not generally acceptable behaviour according to GitHub's terms of service[17]. The authors came to an agreement with GitHub about the duration and use of GitHub's servers, and how the collection should be carried out. GitHub has requested that the details of this agreement should not be published, because it is specific between GitHub and the authors. It is important to emphasise that contacting GitHub before mining is a necessity for ethical research, due to the terms of service. This extends (beyond cloning from GitHub) to using the GitHub API.

7.4 Future Work

There are several areas which would be worthwhile exploring, but could not be explored during the thesis. As the validation of PHANTOM has shown, meaningful clusters could be produced; however, there is still uncertainty on how well this generalises to other repositories. A manual investigation on a random sample of repositories from the large dataset would clarify how well COYOTE works, and how the concept of well-engineered is understood by the tool. This would help to understand why COYOTE was more optimistic about the percentage of well-engineered repositories than reaper. The meaning of the produced clusters could be further analysed by cross-referencing COYOTE's labels with reaper's labels to see where they disagree. Identifying the characteristics of repositories in such cases would deepen the understanding of what the clusters mean.

As of now, PHANTOM extracts five measures from the Git log which were chosen by the authors; however, they do not exhaust the available information. For example, Git can provide a list of files that changed in each commit and how many lines were added or removed. This would enable a code churn related measure, which does not require static code analysis. As long as the additional measures can be formatted as time-series, the later steps (feature extraction, feature selection, and clustering) of PHANTOM are not affected. It would also be of interest to combine measures to analyse whether this would improve the accuracy.

The extracted feature vector could be extended, as well as the feature selection process (based on correlation threshold) could be experimented with (e.g. greedy forward search). K-means was used to cluster repositories, however, other machine learning algorithms, both supervised and unsupervised (e.g. hierarchical clustering, neural networks, support vector machine), and respective configuration parameters could be experimented with. As the time to fit k-means and to predict repositories is negligible, the focus of such experiments would rather be on the prediction accuracy.

Due to the lack of ground-truths in the MSR field, PHANTOM was applied to one ground-truth only. It would be interesting to investigate PHANTOM's accuracy on other datasets and ground-truths. PHANTOM could also be of interest for other contexts. For example, given a collection of agile repositories, PHANTOM could be used to perform cluster analysis to identify different sub-groups (i.e. clusters) of agile repositories and the characteristics these have.

COYOTE could be improved in speed and user interaction. Currently, the tool requires some practice to be handled correctly. COYOTE must be simplified to encourage its usage by others. There is room for several optimisations that would streamline the process. For instance, the time-series conversion and feature extraction, and cloning the Git repositories could be run in parallel, because the computer's CPU and RAM are not exhausted at runtime. Additionally, other input formats (e.g. JSON, SQL) would be helpful, as well as an easy way for users to swap the machine learning algorithm to encourage experimentation.

There are two ambitious improvements which would provide great value. One is to implement functionality to update the feature vectors of previously analysed repositories, which would enable analysis of how repositories change over time. The other is, to implement a distributed version of PHANTOM; COYOTE could be

run on several machines, and by that distribute the load, which would cut down analysis time linearly. For example, using five machines, all of GitHub (80,000,000 repositories) could be analysed in about half a year (compare section 6.1).

8

Conclusion

Traditionally, software project analysis has been approached in form of case studies where researchers manually analyzed a small set of similar projects. Finding these similar projects was both effort intensive and error-prone. However, since the rise of repository hosting platforms like GitHub, large-scale analysis has become more common as we can use automation to mine and compare repositories. One of the biggest problems with large-scale analysis is the acquisition of desirable repositories. Without an appropriate way of filtering out undesirable repositories, analyses will be skewed.

The problem with the majority of applied filtering techniques when mining software repositories is that they are inaccurate or unproven. Munaiah et al. introduce a new method(*reaper*) that achieves high accuracy, but requires the computing power of a computer cluster, and cannot be applied to all repositories as it is based on static analysis of code. In this thesis, a method called PHANTOM is proposed that improves over existing ones. PHANTOM is able to filter repositories accurately with low hardware requirements. It achieves this by using a time-series representation as input to create feature vectors describing properties of the time-series (e.g. number of peaks). These time-series are based on information from the development history and are transformed to feature vectors that can be used with machine learning algorithms for smarter comparison. In particular, this makes the time-series compatible with a standard k-means algorithm, which is used to cluster the repositories into two groups; well-engineered and not well-engineered.

PHANTOM was able to rediscover a ground-truth of 450 repositories, with the best k-means models achieving up to 100% precision and recall. Upon validation of PHANTOM, the best models achieved up to 87% precision or 94% recall when predicting new repositories. The MCC of the best models was overall positive, with the highest being 0.65. This is competitive to the best supervised classifiers from the baseline study (88% precision, 99% recall). PHANTOM obtained the metadata of 1,786,601 GitHub repositories in 21.5 days, which is over 33% faster than reaper, and reduced the hardware requirements by two orders of magnitude. The authors conclude that 38.33% of the analysed GitHub repositories are well-engineered, compared to 24% reported in the baseline study.

In future work a comparison of PHANTOM and reaper is of interest. Identifying where the two methods disagree would open up the opportunity to clarify the differences between them. An additional worthwhile endeavour is to explore additional clustering techniques, which was not possible during the thesis due to time limitations. We provide the produced data, which includes collections of both Git logs, time-series and feature vectors for 650 repositories from the ground-truth and

1,771,167 Git logs of unknown quality.

Bibliography

[1] Wil M. P. van der Aalst. *Process Mining*. Vol. 5. 2016, pp. 301–317. ISBN: 9783642193453. DOI: 10.1007/978-3-642-19345-3. arXiv: arXiv:1011.1669v3. URL: http://www.springerlink.com/index/10.1007/978-3-642-19345-3.

[2] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. "Time-series clustering - A decade review". In: *Information Systems* 53 (2015), pp. 16–38. ISSN: 03064379. DOI: 10.1016/j.is.2015.04.007. arXiv: 1107.3326. URL: http://dx.doi.org/10.1016/j.is.2015.04.007.

[3] Miltiadis Allamanis and Charles Sutton. "Mining Source Code Repositories at Massive Scale using Language Modeling". In: Iim (2013), pp. 207–216.

[4] Sanghamitra Bandyopadhyay and Sriparna Saha. *Unsupervised Classification*. 2013, p. 259. ISBN: 978-3-642-32450-5. DOI: 10.1007/978-3-642-32451-2. URL: http://link.springer.com/10.1007/978-3-642-32451-2.

[5] Moritz Beller, Georgios Gousios, and Andy Zaidman. "Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 356–367. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.62.

[6] John D. Blischak, Emily R. Davenport, and Greg Wilson. "A Quick Introduction to Version Control with Git and GitHub". In: *PLoS Computational Biology* (2016). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1004668.

[7] Jurgen Cito et al. "An Empirical Analysis of the Docker Container Ecosystem on GitHub". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 323–333. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.67.

[8] Valerio Cosentino, Javier L. Canovas Izquierdo, and Jordi Cabot. "A Systematic Mapping Study of Software Development With GitHub". In: *IEEE Access* (2017). ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2682323.

[9] Houtao Deng et al. "A time series forest for classification and feature extraction". In: *Information Sciences* 239 (2013), pp. 142–153. ISSN: 00200255. DOI: 10.1016/j.ins.2013.02.030. arXiv: arXiv:1302.2277v2.

[10] Philippe Esling and Carlos Agon. "Time-series data mining". In: *ACM Computing Surveys (CSUR)* 45.1 (2012), pp. 1–34. ISSN: 03600300. DOI: 10.1145/2379776.2379788. URL: http://dl.acm.org/citation.cfm?doid=2379776.2379788%7B%5C%%7D5Cnhttp://dl.acm.org/citation.cfm?id=2379788.

[11] Robert Feldt et al. "Supporting software decision meetings: Heatmaps for visualising test and code measurements". In: *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*. 2013. ISBN: 9780769550916. DOI: 10.1109/SEAA.2013.61.

[12] Ben D. Fulcher and Nick S. Jones. "Highly comparative feature-based time-series classification". In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (2014), pp. 3026–3037. ISSN: 10414347. DOI: 10.1109/TKDE.2014.2316504. arXiv: 1401.3531.

[13] Mark Gabel and Zhendong Su. "A study of the uniqueness of source code". In: *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. 2010, p. 147. ISBN: 9781605587912. DOI: 10.1145/1882291.1882315. URL: http://portal.acm.org/citation.cfm?doid=1882291.1882315.

[14] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. "Some from Here, Some from There: Cross-Project Code Reuse in GitHub". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 291–301. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.15.

[15] Git. *Git - pretty-formats Documentation*. 2018. URL: https://git-scm.com/docs/pretty-formats (visited on 06/01/2018).

[16] GitHub. *Features · The right tools for the job*. 2018. URL: https://github.com/features (visited on 05/08/2018).

[17] GitHub. *GitHub Terms of Service - User Documentation*. 2018. URL: https://help.github.com/articles/github-terms-of-service/#c-acceptable-use (visited on 06/01/2018).

[18] GitHub. *Starring | GitHub Developer Guide*. 2018. URL: https://developer.github.com/v3/activity/starring/#list-stargazers (visited on 06/01/2018).

[19] Danielle Gonzalez et al. "A Large-Scale Study on the Usage of Testing Patterns That Address Maintainability Attributes: Patterns for Ease of Modification, Diagnoses, and Comprehension". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 391–401. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.8. arXiv: 1704.08412.

[20] International Software Benchmarking Standards Group. *About ISBSG*. 2018. URL: http://isbsg.org/about-isbsg/ (visited on 05/08/2018).

[21] International Software Benchmarking Standards Group. *Project Data from the ISBSG Repository - ISBSG*. 2018. URL: http://isbsg.org/project-data/ (visited on 05/08/2018).

[22] Chonghui Guo. "Time Series Clustering Based on ICA for Stock Data Analysis". In: (2008), pp. 1–4.

[23] Eirini Kalliamvakou et al. “An in-depth study of the promises and perils of mining GitHub”. In: *Empirical Software Engineering* 21.5 (2016), pp. 2035–2071. ISSN: 15737616. DOI: 10.1007/s10664-015-9393-5. arXiv: 2597073.2597074 [10.1145]. URL: http://dx.doi.org/10.1007/s10664-015-9393-5.

[24] Christian Macho, Shane McIntosh, and Martin Pinzger. “Extracting Build Changes with BUILDDIFF”. In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 368–378. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.65. arXiv: 1703.08527.

[25] Nuthan Munaiah et al. “Curating GitHub for engineered software projects”. In: *Empirical Software Engineering* 22.6 (2017), pp. 3219–3253. ISSN: 15737616. DOI: 10.1007/s10664-017-9512-6.

[26] Jeroen Noten, Josh G.M. Mengerink, and Alexander Serebrenik. “A data set of OCL expressions on GitHub”. In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 531–534. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.52.

[27] Alberto S. Nuñez-Varela et al. “Source code metrics: A systematic mapping study”. In: *Journal of Systems and Software* (2017). ISSN: 01641212. DOI: 10.1016/j.jss.2017.03.044.

[28] Wilfredo Palma and Ebook Central (e-book collection). *Time series analysis*. English. 1st ed. Hoboken, New Jersey: Wiley, 2016. ISBN: 1118634322;9781118634325;

[29] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[30] François Petitjean, Alain Ketterlin, and Pierre Gançarski. “A global averaging method for dynamic time warping, with applications to clustering”. In: *Pattern Recognition* 44.3 (2011), pp. 678–693. ISSN: 00313203. DOI: 10.1016/j.patcog.2010.09.013.

[31] Cor-paul Bezemer Queen. “An exploratory study of the state of practice of performance testing in Java-based open source projects”. In: January (2016), pp. 373–384. DOI: 10.7287/PEERJ.PREPRINTS.2496V1.

[32] Ca Ratanamahatana and E Keogh. “Everything you know about dynamic time warping is wrong”. In: *Third Workshop on Mining Temporal and Sequential Data* (2004), pp. 22–25. ISSN: 00903493. DOI: 10.1097/01.CCM.0000279204.24648.44. URL: http://spoken-number-recognition.googlecode.com/svn/trunk/docs/Dynamic%20time%20warping/DTW%7B%5C_%7Dmyths.pdf.

[33] Thomas Rausch et al. “An Empirical Analysis of Build Failures in the Continuous Integration Workflows of Java-Based Open-Source Software”. In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 345–355. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.54.

[34] Baishakhi Ray et al. “A large-scale study of programming languages and code quality in GitHub”. In: *Communications of the ACM* 60.10 (2017), pp. 91–100. ISSN: 00010782. DOI: 10.1145/3126905. URL: http://dl.acm.org/citation.cfm?doid=3144574.3126905.

[35] Gregorio Robles et al. “An extensive dataset of UML models in GitHub”. In: *IEEE International Working Conference on Mining Software Repositories* (2017), pp. 519–522. ISSN: 21601860. DOI: 10.1109/MSR.2017.48.

[36] Simon Rogers, Mark Girolami, and Ebook Central (e-book collection). *A First Course in Machine Learning, Second Edition*. English. 2nd;Second;2; Milton: CRC Press, 2016. ISBN: 1498738486;9781498738545;9781498738484;1498738540;

[37] Mefta Sadat, Ayse Basar Bener, and Andriy Miranskyy. “Rediscovery datasets: Connecting duplicate reports”. In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 527–530. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.50. arXiv: 1703.06337.

[38] Eddie Antonio Santos and Abram Hindle. “Judging a commit by its cover”. In: *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR ’16*. 2016, pp. 504–507. ISBN: 9781450341868. DOI: 10.1145/2901739.2903493. URL: http://dl.acm.org/citation.cfm?doid=2901739.2903493.

[39] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. “Why We Refactor? Confessions of GitHub Contributors”. In: (2016). DOI: 10.1145/2950290.2950305. arXiv: 1607.02459. URL: http://arxiv.org/abs/1607.02459.

[40] Miroslaw Staron et al. “Identifying implicit architectural dependencies using measures of source code change waves”. In: *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*. 2013. ISBN: 9780769550916. DOI: 10.1109/SEAA.2013.9.

[41] Miroslaw Staron et al. “Measuring and visualizing code stability - A case study at three companies”. In: *Proceedings - Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement, IWSM-MENSURA 2013*. 2013. ISBN: 9780769550787. DOI: 10.1109/IWSM-Mensura.2013.35.

[42] Bogdan Vasilescu et al. “Gender and Tenure Diversity in GitHub Teams”. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI ’15*. 2015, pp. 3789–3798. ISBN: 9781450331456. DOI: 10.1145/2702123.2702549. URL: http://dl.acm.org/citation.cfm?doid=2702123.2702549.

[43] Bogdan Vasilescu et al. “On the variation and specialisation of workload - A case study of the Gnome ecosystem community”. In: *Empirical Software Engineering* 19.4 (2014), pp. 955–1008. ISSN: 15737616. DOI: 10.1007/s10664-013-9244-1.

[44] Zhiyuan Wan et al. “Bug Characteristics in Blockchain Systems: A Large-Scale Empirical Study”. In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 413–424. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.59.

[45] Xiaozhe Wang, Kate Smith, and Rob Hyndman. “Characteristic-based clustering for time series data”. In: *Data Mining and Knowledge Discovery* 13.3 (2006), pp. 335–364. ISSN: 13845810. DOI: 10.1007/s10618-005-0039-x.

[46] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. 2014, p. 493. ISBN: 9781605587196. DOI: 10.1145/1810295.1810446. URL: http://portal.acm.org/citation.cfm?doid=1810295.1810446.

[47] Fiorella Zampetti et al. "How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 334–344. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.2.

[48] Chenguang Zhu et al. "A dataset for dynamic discovery of semantic changes in version controlled software histories". In: *IEEE International Working Conference on Mining Software Repositories*. 2017, pp. 523–526. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.49.

A

Overview of Studies in the Field of MSR

Table A.1: Presents an overview the MSR 2017 studies.

Paper	Filtered on	Dataset	Data Source	Data Type	Language
[44]	Domain, market and venture value.	20	GitHub	Bug Database	Many
[19]	Empty and Forked repository	82477	GitHub, GHTorrent	Source code, metadata	Many
[26]	File type	245	GitHub	OCL files	-
[48]	Apache project	8	GitHub	Metadata	-
[24]	Popularity, Commits	-	GitHub	Metadata	-
[5]	Popularity, Forks, Language	1359	GitHub	Travis CI	-
[33]	Popularity, Builds, Developers, Commits	14	GitHub, Travis CI	Build and Git log	Java
[47]	Popularity	20	GitHub, Travis CI	Build and Git log	Java
[7]	File type, Popularity	38079	GitHub	Docker files	-
[34]	Popularity, Language	850	GitHub	Metadata	17
[43]	Domain	1316	GitHub	Metadata	Java
[14]	Language, Developers, Duration, Commits	8753	GHTorrent	Source code	Java
[3]	Forks	14807	GitHub	Source code	Java
[42]	Calendar time, Commits, Duration	-	GHTorrent, GitHub	Metadata	-
[39]	Popularity, Language, Commits	750	GitHub	Source code	Java
[38]	Size, Commits, Language	<3000	Boa	Metadata	Java

Continued on next page

TableA.1– *Continued from previous page*

Paper	Filtered on	Dataset	Data Source	Data Type	Language
[31]	Popularity, Commits	154	GitHub	Source code, Metadata	Java

B

Features extracted by PHANTOM

Table B.1: Features extracted by PHANTOM

Category	Feature	Description
Duration	Duration	Time interval in weeks between the first and last week of the time-series
Y value	Max y	The highest value
	Max y Pos	The week number of the Max Y
	Mean y	The average value
	Sum y	The sum of all values
	q25	The 25% quantile of values
	q50	The 50% quantile of values
	q75	The 75% quantile of values
	std	The standard deviation of values
Peaks	Peak down	The number of downwards facing peaks
	Peak none	The number of peaks that are neither downwards, nor upwards facing peaks
	Peak up	The number of upwards facing peaks
Time between peaks	Min TBP up Avg TBP up Max TBP up	The time between upwards facing peaks is measured as the number of weeks between two neighbouring peaks.
Amplitude	Min amplitude Avg amplitude Max amplitude	The amplitude is the difference in height between a peak and the previous valley. This value is normalised by dividing it with Max Y.
Positive and negative peak deviation	Min PPD Avg PPD Max PPD	The positive peak deviation (PPD) is the difference between the Mean Y value and the y value of a upwards facing peak.
	Min NPD Avg NPD Max NPD	The negative peak deviation (NPD) is the difference between the Mean Y value the y value of a downwards facing peak.
Positive and negative sequences	Min PS Avg PS Max PS Min NS Avg NS Max NS	A sequence is, when at least two sequential gradients have the same sign. Therefore, the positive (PS) and negative sequences (NS) are numeric values, that count the number of sequential same sign gradients.

Continued on next page

TableB.1– *Continued from previous page*

Category	Feature	Description
Positive and negative gradients	Min PG	Gradients are the difference between two neighboring y values.
	Avg PG	
	Max PG	
	Min NG	
	Avg NG	
	Max NG	
	PG Count	Number of positive gradients (PG)
	NG Count	Number of negative gradients (NG)

C

Desktop Computer Specification

Table C.1: Hardware specification of the desktop computer used in this thesis.

Component	Specification
Processor	Intel i5
Ethernet Connection	1Gbps
HDD	1TB
RAM	16Gb DDR3 RAM
Operating System	Lubuntu 17.10

D

K-Means Configuration

Table D.1: K-means configuration for COYOTE. COYOTE uses the standard configuration from the Python library *Scikit-Learn*[29]

Parameter	Value
Number of Clusters	2
Number of Initialisations	10
Centroid Update Algorithm	k-means++ (Lloyd's algorithm)
Max Iterations	300