PHANTOM: Curating GitHub for engineered software projects using time-series clustering

Peter Pickerill · Heiko Joshua Jungen · Mirosław Ochodek · Miroslaw Staron

Received: date / Accepted: date

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. Unfortunately, the existing filtering methods have not kept up with the growth of existing data sources, such as GitHub, and researchers often rely on quick and dirty techniques to curate datasets.

Objective: The objective of this study is to develop a method capable of filtering large quantities of software projects in a time-efficient way.

Method: This study follows the Design Science Research (DSR) 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 the k-means algorithm.

Results: Using the ground truth from a previous study, PHANTOM was shown to be able to rediscover the ground truth with up to 0.87 Precision or 0.94 Recall, and be able to identify “well-engineered” projects with up to 0.87 Precision and 0.94 Recall on the validation dataset. PHANTOM downloaded and processed the metadata of 1,786,601 GitHub repositories in 21.5 days, which is over 33% faster than a similar study, which used a computer cluster of 200 nodes.

Conclusions: It is possible to use an unsupervised approach to identify well-engineering projects. PHANTOM was shown to be competitive compared to the existing su-
supervised approaches while reducing the hardware requirements by two orders of magnitude.

**Keywords** Mining software repositories · GitHub · Data curation · Curation tools

1 Introduction

Software project analysis used to be performed on small corpora of projects, as seen in industry-based case studies (Feldt et al. (2013), Staron et al. (2013a), Staron et al. (2013b)). However, in the recent years, due to the rise of GitHub\(^1\), researchers have had access to a massive corpus of software projects that can be analysed. Consequently, we see more and more studies in the area of software quality that base their research on mining GitHub repositories. Unfortunately, the quality of GitHubs over 100 million repositories is unclear. It has been shown that most repositories can be considered to be of low quality, and could therefore skew analysis (Kalliamvakou et al. (2016)).

For instance, in the recent editions of 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 repositories (e.g., Cito et al. (2017), Gonzalez et al. (2017), Noten et al. (2017), Sadat et al. (2017), Zhu et al. (2017), Rausch et al. (2017), Macho et al. (2017)). 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 (Munaiah et al. (2017)). It is clear then, that for researchers to make use of the large number of repositories available on GitHub, new filtering methods are required.

Munaiah et al. (2017) 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 (Cosentino et al. (2017)). 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.

With this in mind, in this paper, we propose a new method called PHANTOM (Project History Analysis of Time-Series Method) that use unsupervised learning to distinguish between poorly- and well-engineered projects at large scale, based on their development history, while using commodity hardware. We argue that it is possible to achieve comparable results to Munaiah et al. (2017) (which we refer to as the baseline study) by applying the proposed unsupervised-learning method to filter repositories. Since it is an unsupervised method it could also help the researchers to establish a ground truth by automatically grouping projects having similar characteristics. By simplifying the acquisition of information used for

\(^1\) https://github.com
analysis, using the development history exclusively, it is possible to perform the analysis on commodity hardware. We validate PHANTOM on the datasets published in the baseline study (Munaiah et al. (2017)). The method’s efficacy is shown by using the dataset consisting of 650 labelled repositories and its applicability for large-scale analysis is shown when it is applied on the dataset of over 1.8 Million software repositories.

The structure of this paper is as follows. Section 2 presents and discusses related studies. The problem statement and the design of the validation study are presented in Section 3. Section 4 describes the proposed method, which is further validated in Section 5. We discuss the findings from the validation study in Section 6. Finally, we conclude our findings in Section 7.

2 Related Work

Since 1997, the International Software Benchmarking Standards Group (ISBSG) has been collecting project-artefacts data from industry practitioners through questionnaires. They have collected 8,200 projects so far, covering seven major industry types. Unfortunately, the ISBSG database stores only descriptive information about projects without direct access to source code or project artefacts. Due to the nature of data, it has been mainly used for software effort estimation or productivity benchmarking (González-Ladrón-de Guevara et al. (2016)).

The large-scale projects analyses (including code analysis) are mainly performed on open-source code repositories. The implementations of project analyzers capable of processing such repositories proposed in studies by Gabel and Su (2010) and Munaiah et al. (2017) feature a high-level of automation, yet the resources required (e.g. time, computing power) are large. Despite such difficulties, there are multiple studies that performed code analyses of software projects available on SourceForge or GitHub.

Gabel and Su (2010) conducted a study about the uniqueness of source code within C++, C# and Java applications taken from SourceForge. 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 also time-consuming at large-scale, primarily due to data collection limitations. Cosentino et al. (2017) 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. They state

http://isbsg.org
http://www.sourceforge.com
Gousios (2013), http://ghtorrent.org
https://developer.github.com/v3
that “the GitHub API request limit acts as a barrier to get data from GitHub”, which effects curated datasets (such as GHTorrent) and individual researchers. Many researchers criticise the size and up-to-dateness of services like GHTorrent. Cosentino et al 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 (2017) 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 collected its data and the researchers 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 (2016) 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. Also, they 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 APIs 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 (2017) 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. They claim that although the use of metrics “theoretically can be applied to any language”, in practice, it is complex and tools do not support all languages.

In 2017, Munaiah et al (2017) 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

---

6 Dyer et al (2013), http://boa.cs.iastate.edu
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 et al. (2015) presented a decade review on time-series clustering. 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 (2004), 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.

Wang et al. (2006) proposed a method for time-series clustering on global characteristics. They 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. (2013) 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.

A similar, but much more sophisticated approach to time-series classification is presented in the study Fulcher and Jones (2014). 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. According to Fulcher and Jones, this approach outperforms DTW "despite dramatic dimensionality reduction" of the time series. In some cases, one feature was sufficient to classify time-series with high accuracy.

Esling and Agon (2012) 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", 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 (2008) 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 clustering, they were able to identify 4 distinct patterns despite high levels of noise and dimensionality.
3 Research Methodology

The research conducted in this paper follows the Design Science Research (DSR) methodology (Hevner et al. (2004)). In particular, we decided to follow the guidelines provided by Wieringa (2014).

DSR is a problem-solving paradigm that focuses on creating and evaluating artefacts and solutions for practical purposes. In design science, research follows the engineering cycle—it is conducted iteratively until the objectives are reached. The engineering cycle consists of five steps:

1. Problem investigation — understanding the problem and outlining the steps to solve the problem and evaluate the solution.
2. Treatment design — designing an artefact which is used in the study (e.g., a computer program).
3. Treatment validation — evaluating the artefact in a context similar to the context where it is to be used (e.g., lab environment).
4. Treatment implementation — introducing the artefact into the context where it is to be used (e.g., software development organization).
5. Implementation evaluation — evaluation of the effects of the introduction of the artefact on the context (e.g., check whether the program improved the process of software development).

The first three steps of the engineering cycle are called design cycle. According to Wieringa, design cycle is what is usually performed by researchers when designing an artefact while the remaining two steps (treatment implementation and evaluation) can be done once the artefact is in the hands of its intended users.

The design 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. 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.

In our study, we performed a full design cycle to create an artefact—PHANTOM which is the method to analyse software projects, and the problem context is to filter well-engineered projects from GitHub. Firstly, we investigate the problem and define the requirements for the method by analyzing the limitations of the Reaper tool proposed in the study by Munaiah et al. (2017), which we refer to in this paper as the baseline study. Then we perform a validation of the method by applying it to the datasets provided in the baseline study and comparing the results between PHANTOM and Reaper.

The goal of two remaining steps in the engineering cycle is to apply the treatment to and investigate how it interacts with their real-world context. Treatment evaluation is a scalable process and usually requires multiple applications of the treatment to get a full understanding of its usefulness. Although we do not aim at evaluating of the method in this study, we apply PHANTOM to replicate the task of filtering nearly 1.8 Million real software repositories stored on GitHub in

---

1 Munaiah et al. (2017) defines an engineered software project as "a software project that leverages sound software engineering practices in one or more of its dimensions such as documentation, testing, and project management."
the baseline study, what could be perceived as an initial evaluation of the method in its real-life context.

In the remaining part of this section, we investigate the problem of filtering software repositories that justifies the need for designing PHANTOM. Then, we discuss the validation and evaluation procedures used in this study.

3.1 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 \cite{Munaiah2017}. 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 which uses 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. In \cite{Munaiah2017}, 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 \cite{Munaiah2017}, 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\cite{Munaiah2017}. 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 \cite{Munaiah2017}. 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, we translated the identified problems and deficiencies of the existing solutions into the following requirements for the new method:

- **Req1**: All measures must be extractable from the Git Log.
- **Req2**: Time-series must be characterised by feature vectors accurately.
- **Req3**: The established ground truth can be discovered using an unsupervised learning (without knowing the decision classes).
- **Req4**: The method provides comparable accuracy to supervised methods.
- **Req5**: The method performs well on commodity hardware at large-scale.

3.2 Treatment Validation

In order to validate PHANTOM, we performed a series simulation studies on the datasets published in the baseline study. Munaiah et al (2017) published five datasets. These datasets are formatted as collections of GitHub repository URLs. Four of these datasets (Organisation, Utility, Negative Instances, and Validation) are used as ground truth in the validation of the proposed method, with the fifth being referred to as 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** — it 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** — it contains 150 well-engineered repositories. It defines a well-engineered repository as one with a general-purpose. That is to say, a repository that has value to users other than the developers. The repositories were randomly sampled from 1,857,423 GitHub repositories.
- **Negative Instances** — it 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** — it 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** — it 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.

Since the four ground-truth datasets contained labelled data, they allowed us to validate the PHANTOM’s accuracy and compare it with the accuracy of reaper proposed in the baseline study. While evaluating the accuracy of the methods
we used the popular prediction quality metrics Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC).

When predicting the label of an object and comparing the predicted label to the actual label there are four possible outcomes: the object is correctly classified to a positive class—true positive (TP), the object is falsely classified to a positive class—false positive (FP), the object is correctly recognized as not belonging to the positive class—true negative (TN) and the object is falsely recognized as not belonging to the positive class—false negative (FN). We calculate Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC) by aggregating information about the outcomes for all classified cases.

4 PHANTOM—A Developed Artefact

PHANTOM (Project History Analysis of Time-Series Method) is a software-repositories filtering method that addresses the problems identified in Section 3.1. The process used in PHANTOM is presented in Figure 1 and its steps are explained in the subsequent paragraphs.

The input to the method 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 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 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. These measures are explained in Table 3.

In Table 2, the example Git log is transformed into the five measures. Measures are represented as a regular time-series, which makes their comparison possible;
Table 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).

| Format Type | Hashes | Author Name | Committer Name | Date |
|-------------|--------|-------------|---------------|------|
| %H | %P | %an | %ae | %at |
| GUID | GUID | String | String | UTS |
| %cn | %ce | %ct |
| String | String | UTS |

Table 2 Example Git log, where each row is one commit.

| Hashes | Parent Hashes | Author Name | Committer Name | Date |
|--------|---------------|-------------|---------------|------|
| b575f3 | 82c9f95 | ab | cd | 1519904296 |
| 82c9f95 | efaf9cd | ab | cd | 1519904396 |
| efaf9cd | 703b7b1 | ab | cd | 1519904672 |

Table 3 The measures extracted by PHANTOM.

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

Table 4 Example time-series for the five measures.

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

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). In the feature-based approach, a time-series is represented by a feature vector. This feature vector is fixed-length which makes it compatible with common clustering algorithms. A feature vector consists of a number of values that describe certain aspects of time series. For example,
Fig. 2 Example features that can be extracted from a time-series.

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 Figure 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.

In Figure 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 neighbouring 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.

A complete list of the 42 extracted features is presented in Table 5. The measures are extracted separately, which means that there is one feature vector per measure. In Table 6 a sample of the extracted features is presented. 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. Each repository in the input collection is processed in this way. After this, feature vectors are used in the subsequent steps.

The next step is to select the best subset of features from the feature vector (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). In this study, we divided observations into two clusters and used the standard configuration of k-means in the Python library Scikit-Learn\footnote{Pedregosa et al. (2011), \url{https://scikit-learn.org}}

- number of clusters = 2,
- number of initialisations = 10,
- centroid update algorithm = k-means++ (Lloyd’s algorithm),
- max iterations = 300.

Finally, the fitted model is outputted. A new observation is classified to a cluster by measuring its Euclidean distance to all centroids and assigning it to the cluster represented by the closest one.

Although PHANTOM uses unsupervised models, it is worth to emphasize that the features extracted and preprocessed in steps D and E could be used as an input to supervised models, if the ground truth is available for the considered dataset.

5 Validation Results

We validated PHANTOM against each of the requirements defined in Section 3.1. For the requirements requiring evaluating the accuracy of the proposed method or comparing it with the baseline study (reaper), we based the validation on the datasets and prediction quality measures presented in Section 3.

\[ \text{Amplitude} = \left( \frac{1000}{\text{Max Y}} \right) = 0.25 \]

\[ \text{Max Y} = 4000 \]

Fig. 3 Example features that can be extracted from a time-series.
Table 5 Features extracted by PHANTOM

| 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                                            |
| Peeks    |          |                                                                          |
|          |          |                                                                          |
|          | 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     |          |                                                                          |
|          |          |                                                                          |
|          | Min TBP up| The time between upwards facing peaks is measured as the number of weeks between two neighbouring peaks. |
|          | Avg TBP up|                                                                          |
|          | Max TBP up|                                                                          |
| Amplitude| Min 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. |
|          | Avg amplitude |                                                                          |
|          | Max amplitude |                                                                          |
| Positive and negative peak deviation| Min PPD | The positive peak deviation (PPD) is the difference between the Mean Y value and the y value of a upwards facing peak. |
|          | Avg PPD   |                                                                          |
|          | Max PPD   |                                                                          |
|          | Min NPD   | The negative peak deviation (NPD) is the difference between the Mean Y value the y value of a downwards facing peak. |
|          | Avg NPD   |                                                                          |
|          | Max NPD   |                                                                          |
| Positive and negative sequences| Min PS | 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. |
|          | Avg PS    |                                                                          |
|          | Max PS    |                                                                          |
|          | Min NS    |                                                                          |
|          | Avg NS    |                                                                          |
|          | Max NS    |                                                                          |
| Positive and negative gradients| Min PG | Gradients are the difference between two neighbouring 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)                                        |

Table 6 Example feature vectors for the five measures.

| Measure | 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     |
5.1 All Measures Must Be Extractable From The Git Log (Req1)

The five measures extracted by PHANTOM are Integration Frequency, Commit Frequency, Integrator Frequency, Committer Frequency, and Merge Frequency. Each measure uses different information from 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 (see Table 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.

5.2 Time-Series Must Be Characterised By Feature Vectors Accurately (Req2)

Feature vectors must capture the characteristics of time-series accurately to allow k-means finding meaningful clusters. It can be difficult to select features that do this. This problem is illustrated by the two integration frequencies plotted in Figure 4, which were selected from an investigation of twenty repositories using PHANTOM. 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 7). 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 described in Table 5 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,
Table 7 Example feature vector for the time-series in Figure 4 (the values have been rounded).

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

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.

5.3 The Established Ground Truth Can Be Discovered Using Unsupervised Learning (Req3)

We used PHANTOM to fit k-means models on the ground-truth datasets. These are the organisation and utility datasets, which are both complemented with the negative instances so they 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, we explored 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 were 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.

The comparisons of obtained prediction quality measures of Organisation and Utility datasets depending on the correlation threshold are presented in Figure 5. On the Organisation dataset there are many models that achieve Precision and Recall close to 1.0. On the utility dataset, the accuracy is lower with Precision and Recall of up to 0.9. The high Precision and Recall indicate that the models were able to rediscover the majority of true labels for both datasets. Overall, the Organisation repositories could be rediscovered with higher accuracy than the 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 The Method Provides Comparable Accuracy To Supervised Methods (Req4)

In the baseline study, a custom Score-based and Random Forest classifiers were trained on the Organisation and Utility ground-truth datasets (each combined with the instances from the Negative Instances dataset). The classifiers were then used to predict the Validation dataset. In order to compare k-means to these algorithms, PHANTOM is applied to the same datasets. First, similarly to Section 5.3 PHANTOM explores a range of thresholds for each combination of datasets and measures. This time, however, the fitted models are used to predict repositories from the Validation dataset. Accuracy when predicting repositories is shown in Figure 6. As already seen in Figure 5, 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 correlation threshold.

These rules are implemented in PHANTOM which automates the feature selection process. The best models are presented in tables 8 and 9.

The accuracy of the models proposed in the baseline study is presented in Table 10. The baseline classifiers, trained on the Organisation dataset, achieved lower F-measure than any PHANTOM model fitted to the same data. PHANTOM matches 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. PHANTOM matches the F-measure of the score-based classifier. Although the highest Precision on the Utility dataset of 0.82...
### Table 8 Prediction accuracy for the five measures when clustering repositories. (Organisation models)

| 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 9 Prediction accuracy for the five measures when clustering repositories. (Utility models)

| 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 10 The accuracy of the baseline’s tool reaper as reported in [Munaiah et al. (2017)](Munaiah et al. (2017)).

| 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      |

could be surpassed by two out of five PHANTOM models (Integrations, Commits), the Recall on these was lower than the one obtained for the RF classifier.

Finally, we decided to train a set of Random Forest classifiers using the measures and extracted features provided by PHANTOM to investigate whether the features allow obtaining similar results to the baseline study. The prediction accuracy of the RF Classifiers are presented in tables 11 and 12. The F1-Measure scores obtained for the best RF models for the Organisation and Utility datasets were equal to 0.72 and 0.79, respectively. For the Organisation dataset, F1-Measure was higher by 0.15 when compared to the corresponding RF models in the baseline study, while for the second dataset, it was lower by 0.05. This confirms that the features extracted by PHANTOM allow separating instances of well- and not well-engineered projects for both unsupervised and supervised models. Nevertheless, when compared to the accuracy of PHANTOM, the F1-Measure scores for PHANTOM were higher by 0.11 for the first dataset and lower by 0.01 for the second one than for the RF classifiers trained using the same features. Thus, it seems that the unsupervised model used by PHANTOM can achieve higher accuracy prediction than RF for some cases.

---

9 We used the default configuration of Random Forest provided by the Scikit-Learn library since [Munaiah et al. (2017)](Munaiah et al. (2017)) did not provide information about the RF meta-parameters they used in their study.
Table 11 Prediction accuracy for the five measures and Random Forest classifiers. (Organisation models)

| Measure | Precision | Recall | F-Measure | MCC |
|---------|-----------|--------|-----------|-----|
| Merges  | 0.95      | 0.58   | 0.72      | 0.59|
| Integrators | 0.94    | 0.44   | 0.60      | 0.48|
| Integrations | 0.94   | 0.47   | 0.63      | 0.51|
| Commits  | 0.93      | 0.38   | 0.54      | 0.43|
| Committers | 0.93   | 0.42   | 0.58      | 0.46|

Table 12 Prediction accuracy for the five measures and Random Forest classifiers. (Utility models)

| Measure | Precision | Recall | F-Measure | MCC |
|---------|-----------|--------|-----------|-----|
| Merges  | 0.74      | 0.78   | 0.76      | 0.51|
| Integrators | 0.78    | 0.75   | 0.77      | 0.54|
| Integrations | 0.78   | 0.80   | 0.79      | 0.57|
| Commits  | 0.77      | 0.79   | 0.78      | 0.56|
| Committers | 0.80   | 0.75   | 0.77      | 0.56|

Table 13 Number of available repositories and timings to clone and generate the Git log for them.

| Dataset                | Available Repositories | Time Taken     |
|------------------------|------------------------|----------------|
| Organisation           | 149 / 150              | 10:39 minutes  |
| Utility                | 145 / 150              | 2:44 minutes   |
| Negative Instances     | 138 / 150              | 1:53 minutes   |
| Validation(Well-engineered) | 100 / 100          | 1:36 minutes   |
| Validation(Not well-engineered) | 98 / 100          | 1:18 minutes   |
| Large                  | 1,780,773 / 1,857,423  | 21.5 days      |

5.5 The method Performs Well On Commodity Hardware At Large-Scale (Req5)

In order to validate the applicability of PHANTOM to process large-scale data, we performed two analyses. In the first one, we used PHANTOM to download and process the labelled, smaller datasets Utility, Negative Instances and Validation and extrapolate the results to the size of the Large dataset. Since the Organisation dataset comes from a different selection process than the other datasets, it is not used for this extrapolation. The second analysis is performed directly on the Large dataset.

The PHANTOM’s results for both analyses are presented in Table 13. For the smaller datasets, it shows 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, which gives the average repository-download time of 0.04 seconds. When these values are extrapolated up to the Large dataset (1,857,423 software project repositories), the time to obtain the Git logs is estimated to be 20.2 days.

For the Large dataset, it took PHANTOM 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 con-
Table 14 Results from PHANTOM 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     | 0.9       | 343,818                   | 19%                           |
| Commits    | 0.9       | 704,995                   | 4%                            |
| Committers | 0.75      | 688,260                   | 39%                           |
| Integrations | 0.9    | 700,501                   | 39%                           |
| Integrators | 0.75      | 688,260                   | 38%                           |

Table 15 Results from PHANTOM on 1,771,167 repositories from the Large dataset. Each row represents one model (Utility models).

| Measure    | Threshold | Number of well-engineered | Percentage of well-engineered |
|------------|-----------|---------------------------|-------------------------------|
| Merges     | 0.85      | 1,716,118                 | 96%                           |
| Commits    | 0.95      | 985,213                   | 55%                           |
| Committers | 0.75      | 690,371                   | 39%                           |
| Integrations | 0.95    | 983,553                   | 55%                           |
| Integrators | 0.45      | 617,042                   | 35%                           |

verting 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.

Also, we selected the models that performed best on the Validation dataset and applied them to classify the projects in the Large dataset. This means that ten models (one for each combination of ground-truth dataset and measure) were used to predict the repository labels for the Large dataset. In tables 14 and 15, 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 in 19% and 96%. By taking the average of the six closest models, we could predict the percentage of well-engineered projects to be around 38%. The same result was obtained when the percentage of well-engineered projects was predicted by averaging the results of the best-performing models (models trained using the integrations and commits measures; both obtained the highest F-Measure scores of 0.83 and 0.78 depending on the training datasets). This is visibly higher than the prediction for the baseline study, which was around 24%.

6 Discussion

PHANTOM extracts five measures that have a form of time series. Then, 42 features are extracted that characterize each of the time series. The measurement and feature extraction process are done directly on the Git log without the need of accessing any external sources of information (e.g., bug trackers, continuous
integration servers) or analyzing project artefacts (e.g., project documentation or source code).

Also, the validation shows that PHANTOM was able to rediscover the ground truth of the software repository datasets published in the baseline study by [Mu-naiah et al. (2017)], in which software projects were manually labelled. Many of the PHANTOM (unsupervised) models were able to achieve Precision and Recall close to 1.0. That confirms that the measures and extracted features are descriptive enough to allow differentiating between well- and not well-engineered software projects.

By comparing the prediction accuracy of the PHANTOM models presented in tables 8 and 9 with the accuracy of the classifier in the baseline study presented in Table 10, it is clear that the PHANTOM models can compete with the supervised approaches. Committers model achieved slightly higher F-Measure and Recall for the Organization study than the best classifier in the baseline study, and higher Precision for the Utility dataset. This shows, that k-means is a competitive alternative to supervised algorithms for the considered problem. When the features extracted by PHANTOM were used to train supervised models (Random Forest), they obtained similar (and even slightly better) accuracy than the corresponding models in the baseline study.

Also, PHANTOM 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, PHANTOM 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 PHANTOM 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.

PHANTOMs working assumption is that the programming language of a repository is not relevant. As the Git log is independent of the programming language, it can analyse any programming language, which is a significant improvement over reaper. However, the authors must admit that the efficacy of PHANTOM 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.

PHANTOM 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.

PHANTOM reduced the hardware requirements over reaper by two orders of magnitude. Reaper analysed the Large dataset using a computer cluster of 200 nodes, while PHANTOM achieved the same using a desktop computer.

\[10\] Computer hardware and operating system used in this study: Intel i5 CPU, 1Gbps Ethernet, 1TB HDD, 16GB DDR3 RAM, Lubuntu 17.10.
Table 16 Comparison between reaper and PHANTOM.

| 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,      | Any                          |
|                                    | Python, Ruby                | Python, Rust                 |

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

Taking into account the reduction in requirements on both processing time and hardware with respect to reaper that used a cluster of computers, we can conclude that PHANTOM can perform well on commodity hardware, even at large scale.

Interestingly, when both PHANTOM and reaper are applied to the Large dataset (unlabelled data) the obtained result differ visibly. By taking the average results for PHANTOM, we could state that it indicated 38% of projects in the Large dataset as well-engineered while the corresponding number for the reaper was lower and equal to 24%. Therefore, it seems that more research is needed before we can reliably estimate how many well-engineered projects are in big software repositories such as GitHub.

To summarise the discussion, we present a side-by-side comparison between PHANTOM and reaper in Table 16.

6.1 Threats to Validity

Constructs Validity The baseline study defines “engineered software project” as one that leverages sound software engineering practices in one or more of its dimensions such as documentation, testing, and project management. Unfortunately, as Munaiah et al. explicitly state, this definition is intentionally abstract. Consequently, it is not clear what a “sound practice” means. Is the practice sound when one performs certain activities or whether these activities should lead to obtaining specific outputs? In our study, we use five measures (Integration Frequency, Commit Frequency, Integrator Frequency, Committer Frequency, and Merge Frequency) that are taken from Git logs and have the form of time series (measurement performed in time). Therefore, they could reflect the ways of working in the
projects, also revealing some periodic behaviour. However, none of these measures can be used to evaluate the quality of artefacts (e.g., quality of code or software documentation).

**Internal Validity** These five measures we use in the study have been selected by the authors and may not provide full characteristics of the 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.

**External Validity** The ground truth of the dataset published by Munaiah et al. (2017), 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, PHANTOM uses an unsupervised model that was able to rediscover the ground truth by using only the features space. 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 PHANTOM and the ground truth are wrong and this agreement is coincidental. PHANTOM 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.

The predicted number of well-engineered projects in the Large dataset differs visibly between PHANTOM models. A similar observation was made in the baseline study, for which the predicted percentage of well-engineered projects for the Large dataset ranged between 6% and 70%. Therefore, the predictions provided by both studies (as the predictions of the best performing classifiers) should be taken with caution since the level of uncertainty is high.

### 6.2 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 GitHub (2018). 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 Conclusions

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 (2017) 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).”

The problem with the majority of applied filtering techniques when mining software repositories is that they are inaccurate or unproven. Munaiah et al (2017) introduced 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 study, we proposed a new method called PHANTOM that 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 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. As a result, PHANTOM allows researchers to automatically and inexpensively curate desirable repositories for more specific analyses.

In the performed validation, PHANTOM was able to rediscover a ground truth of 450 repositories, with the best k-means models achieving up to 1.0 Precision and Recall. When applied to new, unseen data, the best models in PHANTOM achieved up to 0.87 Precision or 0.94 Recall. 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 by Munaiah et al (2017) that reported 0.88 Precision and 0.99 Recall on the same datasets. 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.

Because of the limitations of existing curating methods, many studies that mine software repository use inaccurate or unproven filtering approaches (e.g., popularity). PHANTOM could be applied in such studies to improve the data curation process. For example, Robles et al (2017) 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.
References

Aghabozorgi S, Seyed Shirkhorshidi A, Ying Wah T (2015) Time-series clustering - A decade review. Information Systems 53:16–38, DOI 10.1016/j.is.2015.04.007, URL http://dx.doi.org/10.1016/j.is.2015.04.007

Cito J, Schermann G, Witten JE, Leitner P, Zumberi S, Gall HC (2017) An Empirical Analysis of the Docker Container Ecosystem on GitHub. In: IEEE International Working Conference on Mining Software Repositories, pp 323–333, DOI 10.1109/MSR.2017.67

Cosentino V, Canovas Izquierdo JL, Cabot J (2017) A Systematic Mapping Study of Software Development With GitHub. IEEE Access DOI 10.1109/ACCESS.2017.2682321

Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Information Sciences 239:142–153, DOI 10.1016/j.ins.2013.02.030, arXiv:1302.2277v2

Dyer R, Nguyen HA, Rajan H, Nguyen TN (2013) Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp 422–431

Esling P, Agon C (2012) Time-series data mining. ACM Computing Surveys (CSUR) 45(1):1–34, DOI 10.1145/2379776.2379788, URL http://dl.acm.org/citation.cfm?doid=2379776.2379788

Feldk R, Staron M, Hütt E, Lilijegren T (2013) 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, DOI 10.1109/SEAA.2013.61

Fulcher BD, Jones NS (2014) Highly comparative feature-based time-series classification. IEEE Transactions on Knowledge and Data Engineering 26(12):3026–3037, DOI 10.1109/TKDE.2014.2316304, 1401.3531

Gabel M, Su Z (2010) A study of the uniqueness of source code. In: Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE ’10, p 147, DOI 10.1145/1882291.1882315, URL http://portal.acm.org/citation.cfm?doid=1882291.1882315

Gonzalez D, Santos JC, Popovich A, Mirakhorli M, Nagappan M (2017) 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, pp 391–401, DOI 10.1109/MSR.2017.8, 1704.08412

Gousios G (2013) The ghtorrent dataset and tool suite. In: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, Piscataway, NJ, USA, MSR ’13, pp 233–236, URL http://dl.acm.org/citation.cfm?id=2487085.2487132

González-Ladrón-de Guevara F, Fernández-Diego M, Lokan C (2016) The usage of ISBSG data fields in software effort estimation: A systematic mapping study. Journal of Systems and Software 113:188–215

Guo C (2008) Time Series Clustering Based on ICA for Stock Data Analysis pp 1–4

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS quarterly 28(1):75–105

Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2016) An in-depth study of the promises and perils of mining GitHub. Empirical Software Engineering 21(5):2035–2071, DOI 10.1007/s10664-015-9393-5, URL http://dx.doi.org/10.1007/s10664-015-9393-5

Machón C, McIntosh S, Pinzger M (2017) Extracting Build Changes with BUILDDIFF. In: IEEE International Working Conference on Mining Software Repositories, pp 368–378, DOI 10.1109/MSR.2017.65, 1703.05827

Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating GitHub for engineered software projects. Empirical Software Engineering 22(6):3219–3253, DOI 10.1007/s10664-017-9512-6

Noten J, Mengerink JG, Screven A (2017) A data set of OCL expressions on GitHub. In: IEEE International Conference on Mining Software Repositories, pp 531–534, DOI 10.1109/MSR.2017.52
Nuñez-Varela AS, Pérez-Gonzalez HG, Martínez-Perez FE, Soubervielle-Montalvo C (2017) Source code metrics: A systematic mapping study. Journal of Systems and Software DOI 10.1016/j.jss.2017.03.044

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830

Ratanamahatana C, Keogh E (2004) Everything you know about dynamic time warping is wrong. Third Workshop on Mining Temporal and Sequential Data pp 22–25, DOI 10.1007/10703.06337

Rausch T, Hummer W, Leitner P, Schulte S (2017) 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, pp 345–355, DOI 10.1109/MSR.2017.54

Robles G, Ho-Quang T, Hebig R, Chaudron MR, Fernandez MA (2017) An extensive dataset of UML models in GitHub. IEEE International Working Conference on Mining Software Repositories pp 519–522, DOI 10.1109/MSR.2017.48

Sadat M, Bener AB, Miransky A (2017) Rediscovery datasets: Connecting duplicate reports. In: IEEE International Working Conference on Mining Software Repositories, pp 527–530, DOI 10.1109/MSR.2017.50

Staron M, Hansson J, Feldt R, Meding W, Henriksson A, Nilsson S, Höglund C (2013a) 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, DOI 10.1109/IWSM- Mensura.2013.35

Staron M, Meding W, Hoglund C, Eriksson P, Nilsson J, Hansson J (2013b) 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, DOI 10.1109/SEA.2013.9

Wang X, Smith K, Hyndman R (2006) Characteristic-based clustering for time series data. Data Mining and Knowledge Discovery 13(3):335–364, DOI 10.1007/s10618-005-0039-x

Wieringa R (2014) Design Science Methodology for Information Systems and Software Engineering. DOI 10.1145/1810295.1810446, URL http://portal.acm.org/citation.cfm?doid=1810295.1810446

Zhu C, Li Y, Rubin J, Chechik M (2017) A dataset for dynamic discovery of semantic changes in version controlled software histories. In: IEEE International Working Conference on Mining Software Repositories, pp 523–526, DOI 10.1109/MSR.2017.49