Retrospective Project Analysis Using the Expectation-Maximization Clustering Algorithm

Steffen Herbold, Jens Grabowski, Stephan Waack
Institute of Computer Science
Georg-August-Universität Göttingen, Germany
Email: {herbold, grabowski, waack}@cs.uni-goettingen.de

***Abstract*—Schedule slips are often the reason for failed projects or low-quality software. Therefore, investigation if a project was on schedule is an important task when analyzing software projects in retrospective. In this paper, we present a data-driven approach for the retrospective determination of project phases through a clustering algorithm. The analysis is based on software metrics measured at different points of time during the project execution. We will describe how the data can be collected, prepared and analyzed. Our findings are validated through a case study where we analyzed two large-scale open-source projects. The results show that it is possible to successfully identify the final phase of a project using our approach.**

***Keywords*-EM clustering, project analysis, repository mining**

I. Introduction

One of the biggest challenges of software projects other than the task at hand is the project plan. Often, time seems too short and schedule slips occur, or features have to be removed to remain on schedule with milestones and release candidates. On the other hand, this can also cause developers to ignore parts of the schedule, e.g., they add features after a feature freeze, instead of focussing on stabilizing the project. This increases the risk of producing low-quality software, which will reduce costumer satisfaction and increase the maintainance costs. It is the task of good software development processes to prevent this. To improve the current process, the retrospective analysis of past projects with respect to their schedule is an important means. This is often performed using experts intuition, based on tangible data consciously and unconsciously taken into account. This can include knowledge about the project environment as well as information about the project itself, e.g., the size or the number of unresolved bugs.

In our previous work [1], we have successfully used the k-means algorithm [2] to identify feature freezes during a project. The approach is based on software metric data about the project's milestones mined from the projects repository. The contribution of this paper is an extension of this approach. First, we extend the points of time for the measurement from milestones to arbritrary points of time $t_1, \ldots, t_n$, e.g., the milestones but also weekly measurements. This allows steering of the coarseness of the analysis. For the clustering, we propose the Expectation-Maximization (EM) clustering algorithm [3], because it is more powerful than k-means. The assumption of the approach is that metric values are similar if they are measured during the same phase of a project and different, if they are from different phases. For that reason, we use clusters as indicators about the actual project phases and when they changed.

The remainder of this paper is structured as follows. In Section II, we introduce software metrics and discuss which metrics we use and how we selected them. Afterwards, in Section III, we explain the basic concept of the EM clustering algorithm. Section IV discusses our methodology for data collection, preparation, and analysis. In Section V, we apply our approach in a case study. The results of our work and the threats to its validity are discussed in VI. Finally, Section VII concludes the paper and gives an outlook on future work on this subject.

II. Software Metrics

The first problem when analyzing software or software projects is that software is an abstract and difficult to grasp product. Software metrics are a means to describe the abstract product software with numbers. The IEEE defines software metrics as "the quantitative measure of the degree to which a system, component, or process possesses a given attribute" [4]. For our analysis, we need quantifiable attributes of software development projects, which is exactly what software metrics can provide.

We use a target-oriented approach for the selection of software metrics, the *Goal/Question/Metric* (GQM) approach [5], [6]. In this approach, first a *goal* that shall be achieved is defined. Then, *questions* are formulated whose answer can be used to achieve the goal. Finally, *metrics* that can answer the questions are selected. This methodology ensures that there is no 'measurement for the sake of measurement', but that it is clear why the metric data is collected and how it is used.

We applied the GQM approach to select metrics to achieve our goal, the *detection of project phases* (Figure 1). We defined two questions to evaluate the goal.

1) How large is the source code?
2) How many bugs are in the software?

 ISBN: 978-1-61208-168-7

Goal	Questions	Metrics
Detection of Project Phases	How large is the source code?	Lines of Code (LOC)
	How many bugs are in the software?	Number of Bugs (BUG)
		Number of Active Bugs (ACTBUG)

Figure 1. GQM approach to select apropriate metrics.

The rational behind these questions is that we feel that the most important features to determine the progress of a project are the software's size and its number of bugs. The size determines how much of the software's source code is written and should increase continously as the projects progresses. The number of bugs is an indicator for the stability of a project. It should drop sharply at the end of the project, as the focus switches from development to stabilizing the project. For the first question, we selected the metric *Lines of Code* (LOC) to measure the size of the software. For the second question, we determined two similar candidates: *Number of Bugs* (BUG) and *Number of Active Bugs* (ACTBUG). The metric BUG measures how many of the total number of bugs known about the software at the end of the project are still open, i.e., have not yet been fixed. The metric ACTBUG includes when a bug has been reported, i.e., it does not measure the number of open bugs with relation to the end of the project, but to the currently known bugs. To our mind, one of these two metrics should be sufficient, because it can be shown that a small number of metrics often performs similar to larger sets [7]. As part of the case studies we plan to investigate this further and evaluate if either BUG or ACTBUG performs better than the other.

III. The EM algorithm

To analyze the data, we use the *EM clustering algorithm* [3], which belongs to the *unsupervised* learning algorithms. Unsupervised means that no prior knowledge except the data is used. In our case, this means that the algorithm does not know when a data point has been measured and which project phase it belongs to, only the metric data itself is known. Clustering algorithms estimate the data sources that created the data. In our case, the data sources are project phases. The EM algorithm determines a mixture of gaussian distributions that fits the data. This mixture is basically a number of k gaussian distributions and each data point "belongs" to the distribution which generated it with the highest probability. Each of the distribitions defines a *cluster*, i.e., a set of points that the algorithm determines to be generated by the same data source. The points are assigned to the clusters based on the likelyhood that the underlying gaussian distribution generated the data point. The number k of clusters is not fixed, but determined by the algorithm itself.

The acronym EM stands for *expectation maximization* and describes the two basic steps of the algorithm: 1) calculate the *expected* likelihood of the current hypothesis; 2) determine a new hypothesis to *maximize* the likelihood. Additional details of the algorithm can be found in [8].

IV. Approach

Our approach for the retrospective analysis of software project consists of three phases: 1) data collection; 2) data preparation; 3) data analysis.

A. Data Collection

We selected the metrics LOC, BUG, and ACTBUG for the evaluation (see Section II). For the analysis, we need the values of these metrics at regular points of time $t_1, \ldots, t_n$ during the project. To collect the metric data in retrospective, access to the software project's repository is necessary.

Source code based metrics, like LOC, can be extracted from a *code versioning system*, e.g., *Concurrent Versions System* (CVS) [9], *Subversion* (SVN) [10] or Git [11]. These system allow the access to the whole history of the source code. That way, the state of source code at the times $t_1, \ldots, t_n$ can be accessed. Once the source code is available, we can measure the LOC with any software measurement tool.

The metrics BUG and ACTBUG are gathered from *bug-tracking systems*, e.g., Bugzilla [12]. Bugtracking systems maintain all data related to bugs, i.e., when they were discovered, in which versions of the software they are present, the current state of the bug, a record or its state-changes, and how it was resolved. Possible states of the bugs are, e.g., OPEN and CLOSED: OPEN indicates that a bug is reported and still being worked on; CLOSED means that the bug is resolved. Possible resolutions are, e.g., FIXED, INVALID, and WONTFIX: FIXED means that a bug has been corrected; INVALID means that the entry is not a bug; WONTFIX means that for some reason the bug will not be fixed. Using all these informations, we extract all known bugs for a specific version of a software to measure the metric BUG. By including the information when a bug has been reported, we measure the metric ACTBUG.

The result of the data collection are the metric values for LOC, BUG, and ACTBUG at times $t_1, \ldots, t_n$.

B. Data Preparation

The collected data needs to be prepared for the analysis. To this aim, we *normalize* the data. Normalization means, that we change the scales of the metrics to the interval $[0, 1]$ while keeping the relative distances between the metric values. The normalized metric values $value_{norm}$

are calculated as $value_{norm} = \frac{value - value_{min}}{value_{max} - value_{min}}$, where $value_{min}, value_{max}$ represent the minimal and maximal measured values of the metric.

The reason for the normalization is to reduce the impact of the metric scales. The different scales of the metrics effect the result of the clustering. The scale of LOC is much larger than the scale of BUG and ACTBUG. This difference can give LOC a higher weight than the other two metrics. The important feature for the analysis are not the absolute values but the relative distances to the other values in the project, because the relative distance reflect the progress. Normalization keeps the relative distances, but removes the scale effects, thereby allowing a better data analysis.

As result of the data preparation, we have a the data set $DATA = \{values(t_1), \ldots, values(t_n)\} \subset [0,1]^3$. The notation $values(t_i)$ stands for the value of metrics LOC, BUG and ACTBUG at t_i.

C. Data Analysis

For the data analysis, we use the EM clustering algorithm and apply it to the metric data. The input of the algorithm is only the metric data itself and no information about project phases according to the project plan or even the date of the measured. As result, the algorithm yields k clusters $C_1, \ldots, C_k \subset DATA$. The clusters are a disjoint partition of the input, i.e., $\bigcup_{i=1}^{k} C_i = DATA$ and $C_i \cap C_j = \emptyset$ for all $i, j = 1, \ldots, k$. The number k is not fixed and the algorithm can determine as many or few clusters as required to fit the data.

If the analysis is successfull, the resulting clusters contain time-adjacent data, i.e., $C_i = \{values(t_j), values(t_j + 1), \ldots, values(t_{j+|C_i|})\}$. Such a cluster describes a time-interval $[t_j, t_{j+|C_i|}]$. The time-intervals can then be mapped to the project phases by an expert to gain knowledge about the project. In case the resulting clusters are not time-adjacent, there are to possible conclusions: 1) the approach failed; 2) the project was chaotic. Which of the two is the case needs to be determined by an expert.

V. Case Studies

To validate our approach, we performed a case study where we applied it to two large-scale open source projects. We designed our case study to answer the following two research questions:

- RQ1: Is the approach able to identify project phases?
- RQ2: is either BUG or ACTBUG sufficient or are both required?

In the following, we will describe the case study methodology and data. Then, we will present the results of the experiments. Based on the results, we answer the research questions in Section VI.

A. Methodology and Data

The experiments we performed in this case study are based on data obtained from the development of projects hosted by the Eclipse Foundation [13]. We mined data about the developement of two versions of the Eclipse Platform project [14], the versions 3.2 and 3.3. We excluded the *Standard Widget Toolkit* (SWT) subproject of the platform from our measurements, as it is for the most part independent of the remainder of the project. We obtained the source code from the Eclipse CVS repository [15]. To measure the bug related metrics, we used SQL queries to directly extract the metrics from a database dump of the Eclipse Bugzilla [16] bugtracking system made available to us.

As dates for the measurements we choose weekly intervals on monday mornings. For the Eclipse Platform 3.2 the starting date was 2005-06-27 and the final measurement was 2006-06-26. For the Eclipse Platform 3.3 the start date was 2006-06-26 and the final date 2007-06-26. For the analysis we used Weka's [17] implementation of the EM clustering algorithm, with a maximum of 100 iterations and a dynamic number of clusters.

To answer the research question, we performed three experiments with both software versions respectively. In the first experiment, we used all three metrics as input for the EM clustering algoritm, in the second experiment we used only the metrics LOC and BUG, and in the third experiment we used only the metrics LOC and ACTBUG. The first experiment is to evaluate if the identification of project phases through the clustering works to answer RQ1. The other two experiments evaluate how each of the bug metrics alone performs in order to answer RQ2.

B. Results

The results of the experiments are visualized in figures 2-7. The figures depict the weekly measured data points for the metric used in the experiments and where the clusters are located. Both projects had several milestones and release candidates. MX stands for milestone X, M0 denotes the beginning of the project. RCX denote release candidate X. The number of clusters varies between two and four, depending on the experiment. Each cluster only contains time-adjacent data. While the number of clusters varies, the last cluster found is in all six experiments very similar. The cluster is within 3 weeks of the first release candidate. The clusters before the last one are inconsistent and vary.

VI. Discussion

In this section we discuss the case study results and use them to answer our research questions. Furthermore, we list the threats to the validity of our studies.

A. RQ1: Is the approach able to identify project phases?

All clusters determined in the case study were time-adjacent, thereby providing evidence that cluster analysis is

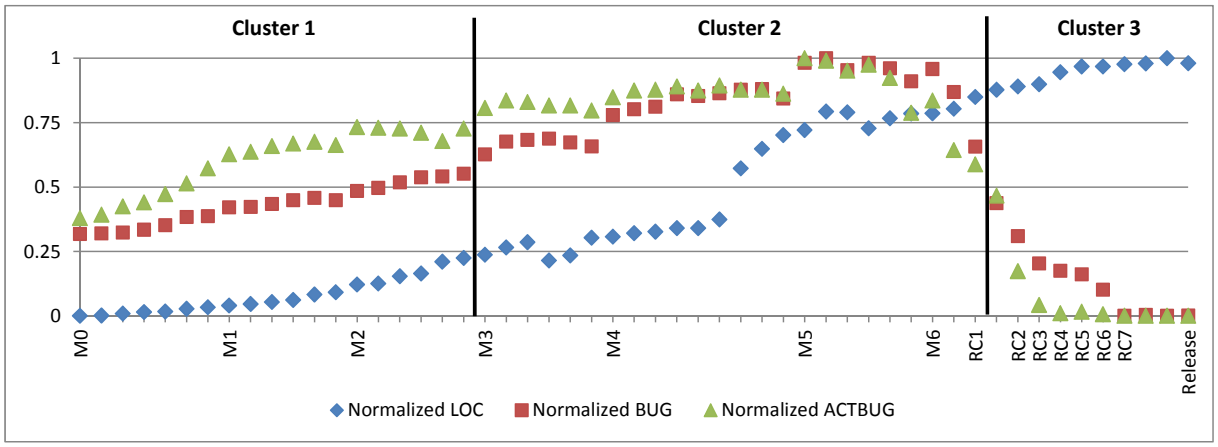

Figure 2. EM clustering with normalized LOC, BUG, and ACTBUG for the Eclipse Platform 3.2

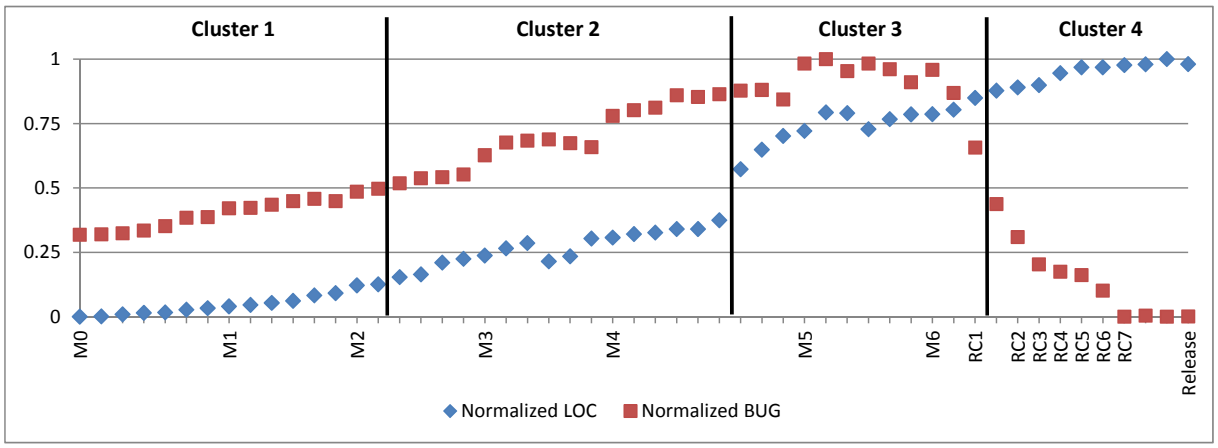

Figure 3. EM clustering with normalized LOC and BUG for the Eclipse Platform 3.2

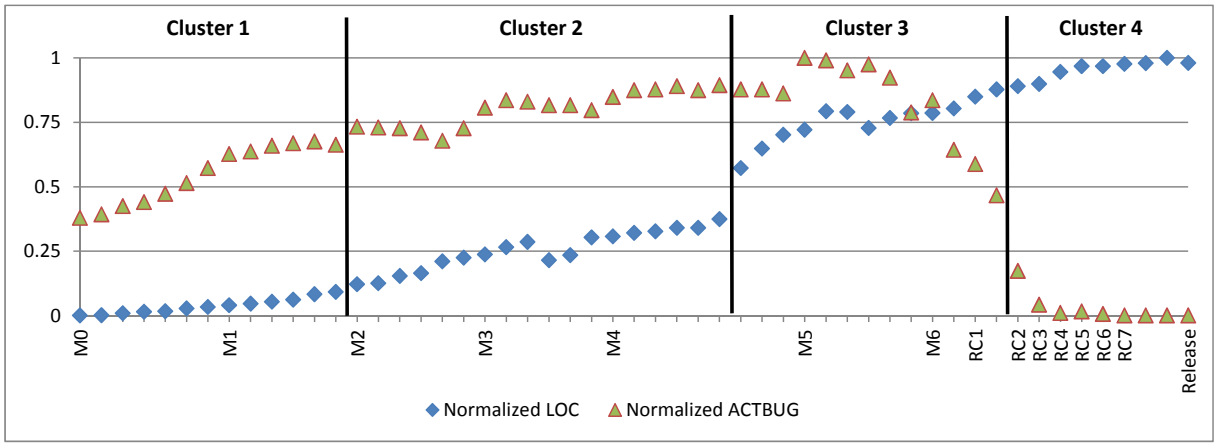

Figure 4. EM clustering with normalized LOC and ACTBUG for the Eclipse Platform 3.2

 ISBN: 978-1-61208-168-7

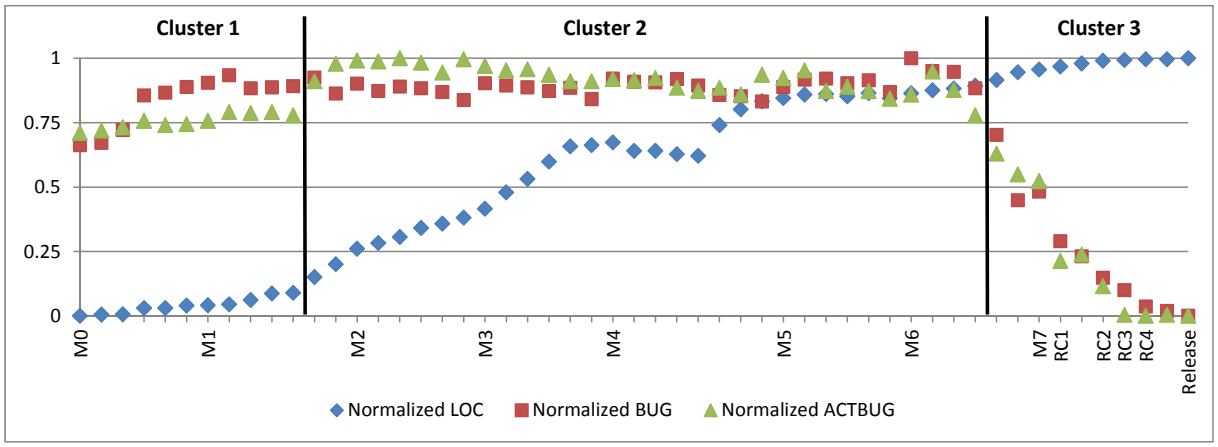

Figure 5. EM clustering with normalized LOC, BUG, and ACTBUG for the Eclipse Platform 3.3

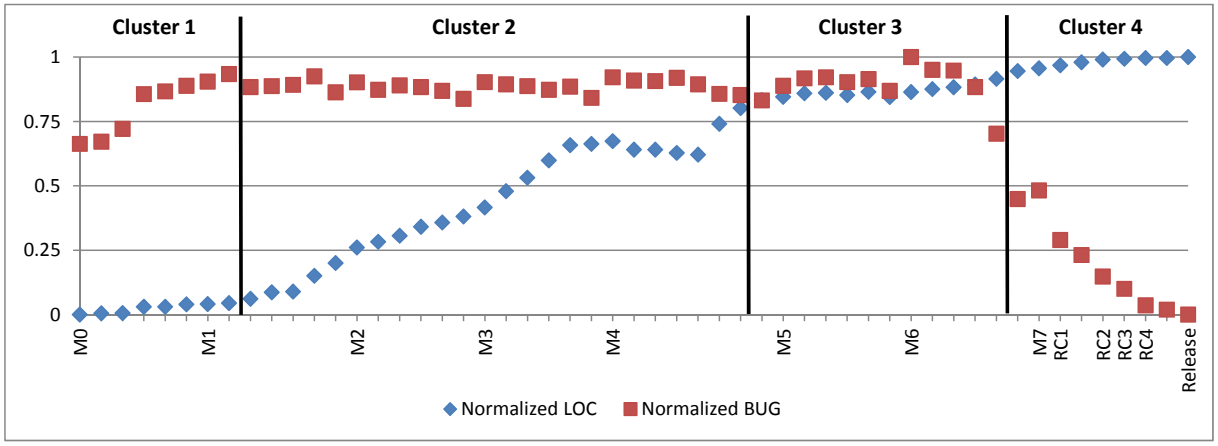

Figure 6. EM clustering with normalized LOC and BUG for the Eclipse Platform 3.3

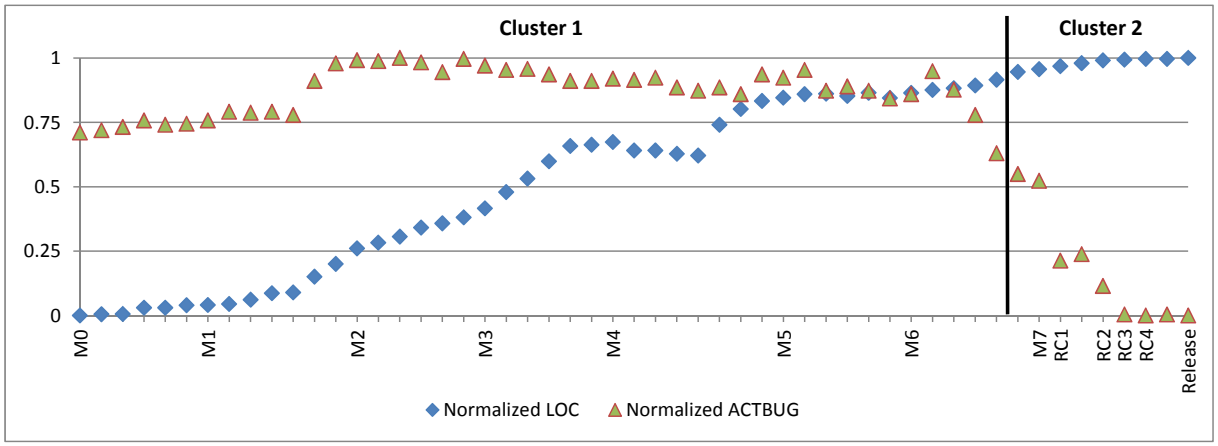

Figure 7. EM clustering with normalized LOC and ACTBUG for the Eclipse Platform 3.3

 ISBN: 978-1-61208-168-7

indead capable of partitioning a project into phases. The case study results show that the final phase of the project leading up to the release can be detected with good accuracy. With regard to this phase, we answer this research question with yes. However, other phases could not accurately be determined and the results varied between the experiments. For example, with the metrics LOC and BUG and for the Eclipse Platform 3.3 (Figure 6), the first cluster seems to be an initializing project phase, where the project was still in planning mode. Thus, this experiment did not only detect the final phase, but also the intial one. But the detection is singular and not repeated accurately in other experiments. Therefore, the approach seems to be capable of detecting further phases, but is not reliable.

B. RQ2: Is either BUG or ACTBUG sufficient or are both required?

When it comes to detecting the final phase of a project, the results show no significant difference between using both BUG and ACTBUG or only one of them. Thus, any of the three combinations is feasible. Therefore, one should use either only BUG or ACTBUG instead of using both, as it reduces the demands on the data mining as well as the dimension of the data set, thereby simplifying the analysis.

C. Threats to validity

There are several threats to the validity of our results. Our case study was only performed for successful industrial open-source projects. We did not consider closed-source projects, or community driven open-source projects. Furthermore, both projects are consecutive version of the same software. The results of the case study are only consistent when it comes to the detection of the final project phase and inconsistent otherwise, indicating possible problems with the analysis.

VII. Conclusion

In this paper, we defined an approach for the retrospective analysis of software development projects. The approach is purely data-driven and based on software metrics. We described how we selected appropriate metrics using the GQM approach. As basis for the analysis we use metric data measured at different times during the execution of a project. We then partition the data into clusters using the EM clustering algorithm. Aim of the analysis is to map the clusters to phases of the project. Our case studies showed that the approach can accurately determine the final phase of a project, but has problem detecting prior phases.

Future work on this project has several promising directions. First, it is possible to tweak the clustering algorithm used for the analysis, e.g., by predefining a number of clusters that matches the project plan. A detailed comparison with other clustering algorithms should also be explored. Second, the metric set can be extended with further metrics. For example, the number of successfull tests or the overall complexity of the project. Third, the time intervals used for the measurement can also be varied to try to determine whether they have an effect on the results. Finally, we will consider further projects to broaden the scope of the case studies.

References

[1] S. Herbold, J. Grabowski, H. Neukirchen, and S. Waack, "Retrospective Analysis of Software Projects using k-Means Clustering," in *Proc. of the 2nd Design for Future 2010 Workshop (DFF 2010), Bad Honnef, Germany*, May 2010.

[2] D. J. MacKay, *Information theory, inference, and learning algorithms*. Cambridge Univ. Press, 2003.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc.*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://www.jstor.org/stable/2984875

[4] IEEE, "Ieee glossary of software engineering terminology, ieee standard 610.12," IEEE, Tech. Rep., 1990.

[5] V. Basili and D. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Softw. Eng.*, vol. 10, no. 6, pp. 728–738, 1984.

[6] V. Basili and H. Rombach, "The TAME project: towards improvement-oriented softwareenvironments," *IEEE Trans. Softw. Eng.*, vol. 14, no. 6, pp. 758–773, 1988.

[7] S. Herbold, J. Grabowski, and S. Waack, "Calculation and Optimization of Thresholds for Sets of Software Metrics," *Empirical Softw. Eng.*, pp. 1–30, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9162-z

[8] T. Mitchell, *Machine Learning (Mcgraw-Hill International Edit)*, 1st ed. McGraw-Hill Education (ISE Editions), Oct. 1997. [Online]. Available: http://www.worldcat.org/isbn/0071154671

[9] July 2011. [Online]. Available: http://www.nongnu.org/cvs/

[10] July 2011. [Online]. Available: http://subversion.apache.org/

[11] July 2011. [Online]. Available: http://git-scm.com/

[12] July 2011. [Online]. Available: http://www.bugzilla.org/

[13] July 2011. [Online]. Available: http://www.eclipse.org/

[14] July 2011. [Online]. Available: http://www.eclipse.org/platform/

[15] July 2011. [Online]. Available: dev.eclipse.org:/cvsroot/eclipse

[16] July 2011. [Online]. Available: https://bugs.eclipse.org/bugs/

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, November 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

 ISBN: 978-1-61208-168-7