Retrospective Analysis of Software Projects using k-Means Clustering

Steffen Herbold[1], Jens Grabowski[1], Helmut Neukirchen[2], Stephan Waack[1]

[1] Institute of Computer Science,
University of Göttingen, Germany

{herbold,grabowski,waack}@cs.uni-goettingen.de

[2] Faculty of Industrial Engineering,
Mechanical Engineering and Computer Science
University of Iceland, Iceland

helmut@hi.is

Abstract

Software projects are usually analyzed by experts based on their previous experience, their intuition and data they gather about the project. In this work, we show an approach for a purely data-driven retrospective project analysis. We plan to build on this work to make predictions about the evolution of software projects.

1 Introduction

Schedule slips are usually discussed as part of software development projects to learn from failures of finished projects to avoid the same failures in future projects. A common reason for schedule slips is that features are still added after a feature freeze instead of stabilizing the project as planned. During a retrospective, the diagnosis whether more than stabilization took place after the feature freeze is usually based on intuition, i.e. tangible data consciously and unconsciously taken into account, e.g., whether the number of open bugs decreased or increased.

To improve the confidence of the developers' intuition, we use data mining [4] and extract information from the repository of a software project and use machine learning to identify a feature freeze. To this aim we gather software metrics [2] about intermediate versions of the software during the development and use the k-means algorithm to cluster the versions into two sets. The one set is interpreted as the one before and the other as the one after the feature freeze. The data is chosen to mimic the intuition of a developer by using the number of open bug reports as indicator of the quality of a software product and the size of the source code as indicator for the project growth. The metric data is mined from two sources: a version-controlled source code repository and a bug tracking system.

For validation of the approach, we present results from a case study where we apply our approach to intermediate versions of projects from the Eclipse[1] ecosystem.

2 Mining

For mining a project in retrospective, we have two requirements on the infrastructure. Firstly, the source code needs to be managed using a version-controlled repository, e.g., CVS[2]. Then, we are able to obtain intermediate versions of the software from the repository to measure the size of the project. As measure for the size, we use the Lines of Code (LOC) of the source code.

Secondly, the project needs to use a bug tracking system, e.g., Bugzilla[3]. The tracking data, i.e. when a bug report was filed and when it was afterwards resolved can be used to determine the Number of Open Bug Reports (BUG) for intermediate versions of the software.

Thus, the mining boils the data contained in the source code repository and the bug tracking system down to tuples (LOC, BUG) for each measured version. The set of all these tuples is interpreted as a subset of the two-dimensional real-space $\mathbb{R}^2$.

3 Analysis

The aim of the analysis is to separate the measured data into two sets containing the versions before and after the feature freeze, respectively. For this purpose, the k-means algorithm, an intuitive and popular clustering algorithm, is used. In the following, we only outline the algorithm, a detailed analysis can be found in [3]. The k-means algorithm yields k clusters that are identified by their centers as their representatives. The algorithm randomly chooses k centers and iterates the following two steps until convergence:

[1]http://www.eclipse.org/
[2]http://www.nongnu.org/cvs/
[3]http://www.bugzilla.org/

1. Each data point is assigned to the cluster represented by the closest center.
2. Each cluster center is replaced by the coordinate-wise average of all points belonging to the corresponding cluster.

These two steps minimize the intra-cluster variance. With respect to our application this means that the versions inside a cluster are somewhat similar. Therefore, under the assumption that versions before the feature freeze are more similar to each other than to those after the feature freeze, the algorithm separates the versions at the feature freeze.

Different metric scales can impact the weight given to each metric by the k-means algorithm during the first step. Large scales lead to greater distances, therefore metrics with large scales are favored. To eliminate the effect that different metric scales have on the resulting clusters, we normalize the metric data. Normalization means that all metric scales are converted to $[0, 1]$ while keeping the relative distances between the values. Another aspect is that normalized data is better suited for inter-project comparisons, as the size is no longer a factor because normalized metrics are relative to the size, but do not include the size anymore.

4 Case Study

To validate our approach, we applied it to two projects. To allow a good comparison of the results, we chose projects that had the same project plan with respect to the produced milestones and release candidates: the Eclipse Platform[4] project 3.2 and the Eclipse Java Developement Tools (JDT)[5] project 3.2. Both projects were mined as described in section 2. We only measured the Java source code of both projects, test code was not included in our measurements. Other sources, such as XML files, were ignored as well. Both projects consist of more than 500 KLOC. In addition to the final version, six milestones and six release candidates were produced in each project. Furthermore, we used version 3.1 as a baseline for the development. Thus, we obtained 14 metric tuples (LOC, BUG) for both projects.

According to the project plan, at Milestone 5, the API was frozen and at Milestone 6, the development was frozen. Therefore, our experiment is successful if the clusters separate the intermediate versions around Milestone 5 or Milestone 6.

When applying the k-means algorithm to the normalized data, the software versions are correctly clustered: for each project, the adjacent versions from the version 3.1 to Milestone 4 are assigned to one cluster, the remaining adjacent versions from Milestone 5 to the final release version are assigned to the second cluster. Thus, our approach correctly detected the

[4]http://www.eclipse.org/platform/
[5]http://www.eclipse.org/jdt/

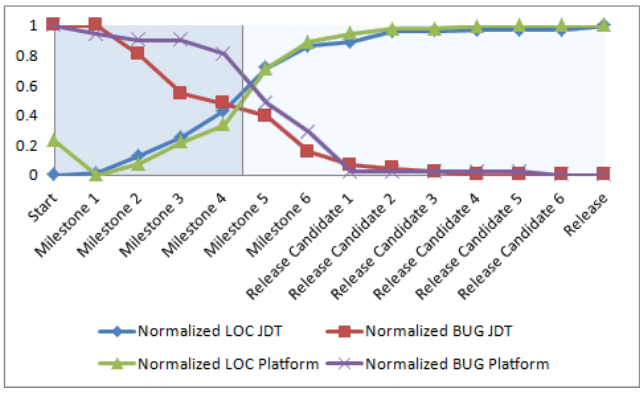

Figure 1: Measured values and cluster assignments

feature freeze between Milestone 4 and Milestone 5 for both projects. Figure 1 visualizes the development of the measured values of both projects, the vertical line indicates the cluster assignments: the area left of the line belongs to one cluster, the area on the right to the other.

5 Future Work

In the future, we plan to apply our approach in further case studies with more projects to increase the confidence into its generality. At the same time, we will investigate additional software metrics that can be used as basis for the analysis. These might include change metrics like *Lines of Code Changed Since the Last Version* or *Number of Fixed Bugs*. Furthermore, more sophisticated clustering techniques like the *EM-algorithm* [1] may be used and their performance will be evaluated against the results produced by the k-means algorithm.

Based on the increased amount of data, we also aim to devise techniques that allow us to draw conclusions not only in retrospective, but rather predictions for the future. To this aim, *supervised* learning will be utilized to train *predictors* about the future of a project. Such predictors can infer information about the probable future of projects, e.g., whether a project is on schedule or not. These predictions can in turn be used by the project management to support steering decisions.

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[2] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 1998.

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

[4] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.