Poster: A changeset-based approach to assess source code density and developer efficacy

Sebastian Hönel
Department of Computer Science
Växjö, Sweden
sebastian.honel@lnu.se

Morgan Ericsson
Department of Computer Science
Växjö, Sweden
morgan.ericsson@lnu.se

Welf Löwe
Linnaeus University
Växjö, Sweden
welf.lowe@lnu.se

Anna Wingkvist
Linnaeus University
Växjö , Sweden
anna.wingkvist@lnu.se

ABSTRACT

The productivity of a (team of) developer(s) can be expressed as a ratio between effort and delivered functionality. Several different estimation models have been proposed. These are based on statistical analysis of real development projects; their accuracy depends on the number and the precision of data points. We propose a data-driven method to automate the generation of precise data points. Functionality is proportional to the code size and Lines of Code (LoC) is a fundamental metric of code size. However, code size and LoC are not well defined as they could include or exclude lines that do not affect the delivered functionality. We present a new approach to measure the density of code in software repositories. We demonstrate how the accuracy of development time spent in relation to delivered code can be improved when basing it on net- instead of the gross-size measurements. We validated our tool by studying ca. 1,650 open-source software projects.

CCS CONCEPTS

• **Software and its engineering** → *Software libraries and repositories*;

KEYWORDS

Software Repositories, Clone Detection, Source code density, Effort estimation

ACM Reference Format:
Sebastian Hönel, Morgan Ericsson, Welf Löwe, and Anna Wingkvist. 2018. Poster: **A changeset-based approach to assess source code density and developer efficacy**. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3195105

1 INTRODUCTION

Gross size of software is effortlessly measured in number of lines of source code. Without determining the net size, estimations related to or based on size will show major deviations. Density is a measure that expresses the ratio between the net- and gross-size of software. Therefore, obtaining a robust measurement of density is important, as it affects any supplementary analyses.

Software cost- and effort-estimation models heavily rely on measuring the size of software. Today, Cocomo II [2] is the de facto industry standard for such estimations. However, often there is not enough data available or the data available is not precise enough to adapt these models, and deviations of orders of magnitude in effort estimations largely reduce their usefulness [7].

Developers have different capabilities, and different developers might approach the same problem in different ways. The time it takes to implement a certain requirement or quantity of functionality thus varies. Many programming languages allow to express a certain functionality in different but semantically equivalent ways. For example, a multi-line `for`-loop can be expressed in one line using functional programming.

Furthermore, unnecessary additions can contribute to inflating the gross size of software without extending the expressed functionality. Comments, whitespace or empty lines, and dead and unreachable code all increase the gross size [8]. Clone detection and other string similarity measures, such as *n-grams* [5], allow us to more precise determine the changes to functionality and thus prevent falsifying effort estimations based on the change of the (gross) size of software, on a per-commit and per-developer basis.

2 RESEARCH GAP

According to [9], "software cost estimation is the process of predicting the effort required to develop a software system". Models such as Cocomo II heavily rely on an assessment of the time to implement a certain functionality, and the size or quantity of that functionality. An empirical validation [6] of popular models for software cost estimation showed that significant calibration was needed. One of those models was COCOMO [1], which has a successor, COCOMO II. [4] indicates that such models are hardly generalizable to environments they were not designed for.

Conducting a sound software cost estimation is a two-fold operation that requires knowledge about the size of the software and the

Table 1: Comparison of the best and worst result obtained

	Best	Worst
Correlation	0.2763	0.0058
Correlation-type	Spearman	Pearson
Session length	6 hours	90 minutes
Similarity measure	*none*	Sorensen-Dice(2)
Clone-detection	no	yes
Remove comments/whitespace	✓	

effort that is required to implement it. While the latter is obtained from the developer's time sheets, it takes accurate measurements of size and functionality to calculate the costs. We introduce the notion of *density*, expressed as $D = F/S$. The effort, when expressed as functionality over time, can also be conveyed as $E = D \times S/t$. ISBSG[1] defines the productivity as effort per size, hence we can deduce the equation $P = D/t$ and demonstrate direct relations of density to effort and productivity.

3 METHOD

In modern software development, software is developed iteratively, by having developers regularly submit (commit) their source code and changes to software repositories. The tool[2] we developed, analyzes each file in such a commit (changeset) and compares it to its previous version. It can operate on *Git*-repositories and considers pure adds or deletes and changed files. The latter type is divided into pure and impure changes.

To obtain a notion of time, we have adapted a tool called *Git-hours*[3]. It allows to group all commits of a developer into sessions. Initial commits of a session are awarded a constant amount of time to account for work done before the first commit. The maximum difference in time between two commits to start a new session is its second parameter. The time spent was analyzed using more than 25 different notions of time, ranging from 30 minutes to 24 hours. More than 1,650 open source projects have been analyzed, approx. 800 Java-, 400 C#- and 400 PHP-projects. Each project had at least 30 commits per developer. Only files containing source code for their respective language were considered during the analysis. Our analysis aimed to cover a great variety of projects that were differently configured.

4 RESULTS AND DISCUSSION

From the amount of data obtained, we have evaluated the best and worst datasets that were used to approximate a certain notion of time by a certain notion of size. Best and worst refer to the correlation between these two dimensions. Measurements were obtained using *Pearson*-, *Spearman*- and *Kendall*-τ-correlations. The best result does neither use clone detection nor string similarities but removes comments and whitespace, and is based on session lengths of six hours. The worst result however uses clone detection, the *Sorensen-Dice* coefficient with shingle-length two, and 90 minutes long sessions (cf. Table 1). Evaluations of differently configured datasets showed that the absence of string similarities or clone detection sometimes improved the correlation, and other times not.

Initially, all time was logged with the default session duration of two hours. This quickly unveiled a shortcoming of *git-hours*: session-initial commits are awarded with a constant amount of time (by default two hours). This leads to a significant number of loggings with this offset, as most commits are farther scattered than two hours. Those loggings were rigorously ignored as they only add noise to the data. An interesting finding is the typical behavior of how developers commit. The second most significant number of loggings is typically within 30 minutes after the session-initial commit and in some cases even within less than a minute. This is due to *bulk-committing*; developers tend to accumulate work and then perform multiple commits within a short amount of time.

We observed major deviations within the various measurements of size we took and were able to identify shortcomings with the tool used for obtaining notions of time. These shortcomings lead to heavy distortions and irregularities in measuring the productivity. As of now we can only conjecture that a regularly logged notion of time will improve the correlation of net-size compared to gross-size.

5 CONCLUSION AND FUTURE WORK

The distortions in time measurement do not currently permit for a more exhaustive analysis of source code density. We will apply our tool to repositories from industry where we also have access to precise time logs. Further, various approaches to determine the similarity between two versions of a software exist. In this research we focus on clone detection and string similarity measures.

As part of future work, we intend to evaluate the collected data points of density against other existing methods (cf. [3]) that are able to extract a notion of effort from a software repository. We will also use quality metrics to establish correlations between those and density.

Collecting the amount of time worked on a commit automatically, solely based on a commit's time stamp, does not accurately reflect the time actually worked. Thus, it is not an adequate notion of time to assess developer efficacy in correlation to delivered functionality, even if we were able to produce more precise notions of size.

REFERENCES

[1] B Boehm et al. 1981. *Software engineering economics.* Vol. 197. Prentice-hall Englewood Cliffs (NJ).
[2] B Boehm, R Madachy, B Steece, et al. 2000. *Software cost estimation with Cocomo II with Cdrom.* Prentice Hall PTR.
[3] G Robles et al. 2014. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories.* ACM, 222–231.
[4] G Finnie and G Wittig. 1996. AI tools for software development effort estimation. In *Software Eng: Education and Practice, Computer Society Press.* IEEE, 346–353.
[5] B Fluri, M Wuersch, M PInzger, and H Gall. 2007. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on software engineering* 33, 11 (2007).
[6] C Kemerer. 1987. An empirical validation of software cost estimation models. *Commun. ACM* 30, 5 (1987), 416–429.
[7] B Kitchenham, D R Jeffery, and C Connaughton. 2007. Misleading metrics and unsound analyses. *IEEE software* 24, 2 (2007).
[8] E Kodhai, S Kanmani, A Kamatchi, R Radhika, and B Vijaya Saranya. 2010. Detection of type-1 and type-2 code clones using textual analysis and metrics. In *Int Conf on Recent Trends in Information, Telecom and Computing.* IEEE, 241–243.
[9] H Leung and Z Fan. 2002. Software cost estimation. In *Handbook of Software Eng and Knowledge Eng: Volume II: Emerging Technologies.* World Scientific, 307–324.

[1] ISBSG. "Software size as the main input parameter to cost estimation model". http://isbsg.org/software-size/, Accessed February 2018
[2] Git-density. "A tool to assess source code density". https://github.com/mrshoenel/git-density
[3] Git-hours. "Estimate time spent on a Git repository". https://github.com/kimmobrunfeldt/git-hours, Accessed February 2018