Measuring Developer Contribution from Software Repository Data

Georgios Gousios   Eirini Kalliamvakou   Diomidis Spinellis

Department of Management Science and Technology
Athens University of Economics and Business
{gousiosg,ikaliam,dds}@aueb.gr

ABSTRACT

Apart from source code, software infrastructures support-
ing agile and distributed software projects contain traces of
developer activity that does not directly affect the product
itself but is important for the development process. We pro-
pose a model that, by combining traditional contribution
metrics with data mined from software repositories, can de-
liver accurate developer contribution measurements. The
model creates clusters of similar projects to extract weights
that are then applied to the actions a developer performed
on project assets to extract a combined measurement of the
developer's contribution. We are currently implementing the
model in the context of a software quality monitoring sys-
tem while we are also validating its components by means
of questionnaires.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Process metrics*;
D.2.9 [**Software Engineering**]: Management—*Productiv-
ity*

General Terms

Measurement, Management

Keywords

Contribution, Software repositories

1. INTRODUCTION

An important aspect of all engineering principles is the
assessment of the contribution of individuals that work on
a project. Contribution assessments are performed to moni-
tor the rate of project development, identify implementation
bottlenecks and isolate exceptional cases, while the results of
contribution assessments can help with future project plan-
ning. A common problem with contribution assessment is
the definition of what contribution is in a particular context

and also the selection and application of the appropriate
measurements.

In software engineering, contribution assessment entails
the measurement of the contribution of a person in terms of
lines of code or function points towards the development of
a software project [7]. In the recent years, the shift to more
agile development practices and the proliferation of software
and project management tools has reduced the estimation
capacity of classic software estimation models. A software
developer today is not only required to write code, but also
to communicate with colleagues effectively and to use a va-
riety of tools that produce and modify code with minimal
input from his side. This change has become more apparent
with the emergence of Open Source Software (OSS).

In this paper, we propose a new model for evaluating de-
veloper contribution, which takes advantage of software de-
velopment repositories to extract contribution indicators.

2. EXISTING WORK

Despite how central the notion of contribution is to soft-
ware development (especially OSS) it is not a central theme
in related literature in itself. Developer contribution is dis-
cussed in several contexts and is often related to other no-
tions such as productivity, participation or activity.

Productivity in abstract terms is the rate of output per
unit of input; it is essentially a measure of production effi-
ciency. Productivity evaluation usually entails extrapolating
a measurable quantity of a unit of work over the resources
required to produce it. In software engineering, productivity
has been a long-studied issue, as it has both managerial and
engineering aspects.

An important, and still open, issue is the definition of
what a unit of work is in the context of software engineering.
In one of the first studies appearing in the literature [11],
Sackman refers to words as the unit of work. The pro-
gramming code is viewed as literature text and each space-
separated identifier is considered to be a unit of thought. As
engineers started to understand that programming language
constructs cannot be classified as intellectual work, the unit
of work definition shifted to lines of code (LOC), which still
remains the most widely used measure of software size. A
weakness of the LOC metric is that it can only be determined
safely at the end of a project. To that end, function points
analysis usually complements the LOC metric as a unit of
work for software estimation. Other units of work that have
been proposed in the literature are classes, methods and
components. A discussion on the various units of work in
the context of object oriented software can be found in [3].

There is no clear definition of what is contribution in the context of software development. The project resources that usually receive input are identified in references [6] and [9]. Software developers participate mainly by writing code, but also by following discussions and submitting comments and ideas to mailing lists as well as performing bug fixes [10, 13]. As a result, the source code repository, the mailing list archives and bug databases are the widely used data sources for discussing participation and activity in a software project.

In reference [4], Glass et al. propose a distinction between clerical and intellectual types of actions in software development. Drawing on contemporary literature, the paper reports extended taxonomies of activities, spanning the software development process from its initial stages to its later ones. Our proposed asset/action taxonomy is in agreement with their lists of software tasks.

Finally, Amor et al. present a model for development effort estimation based on data extracted from software process traces [1]. Their model bears some similarity with our model in the basic idea of using repositories to extract process data, but their focus is on estimation rather than effort and contribution evaluation.

3. OUR APPROACH

Our work is concerned with the measurement of developer involvement and activity in the face of agile and distributed development practices. The model we are building exploits the availability of publicly accessible software repositories to perform measurements and identify possible trends by combining data across projects. It can also run fully automatically with no human intervention.

Our model departs from the classic measurement practices as it does not consider source code as the only contribution metric. This is a deliberate choice that we believe reflects better on how software is developed in an agile context, where an important portion of development time is spent on communication and manipulation of development support tools. Our model does not completely neglect the importance of source code either; we still use the lines of code metric[1] as our basic contribution metric but additionally we also scale it according to a factor that results from the combination of the developer's actions on project assets. For a developer d, the contribution function $C(d)$ is defined as the sum of the total lines of code for the developer $LoC(d)$ plus the contribution factor function $CF(d)$.

$$C(d) = LoC(d) + CF(d) \qquad (1)$$

The important part of our model is the definition of the contribution factor function. To identify which actions can be classified as contribution, we follow a hierarchical, top-down approach: we first identify the project assets that can potentially receive contribution and then analyze the actions that can be performed on each of the identified assets to see if they constitute a contribution or not. The actions are initially identified intuitively and through personal experience; in Section 5, we present the method we will use for validating our selection of actions. Also, not all actions have a positive effect on a project; for example, a commit to a source code

repository that leads to a bug or decreases the quality of the project as it is measured by metrics might be considered a negative contribution. Furthermore, not all actions have the same importance on the evolution of a project; for this reason, we also specify weights that are attached to each action.

In Table 1, we present a non-exhaustive breakdown of actions that can be performed on the identified project assets. Most actions are self-explanatory and relatively easy to mine from each asset repository using simple heuristics or external tools [12]. Each action is a measurable entity whose value is updated after the corresponding project asset has been updated. Values are calculated per developer and are aggregated per project asset. As each project asset might have a different importance for each project, we also assign a weight to the contribution of each asset to the total score for a project, as seen in equation 2. The value A^x denotes the sum of all the events affecting asset x in the project's lifetime.

$$A_{total} = \alpha * A^{rep} + \beta * A^{ml} + \gamma * A^{bug} + \delta * A^{wiki} + \epsilon * A^{irc} \quad (2)$$

The values of the weights $\alpha, \beta, \gamma\delta$ and ϵ are calculated as the percentage of the sum of events for an asset in the total number of events, *across projects*. Individual projects are also allowed to assign project-specific values to the weights to account for what each project deems important for its development process. In the latter case, the value of A_{total} is not of any particular use. As an interesting side effect however, in the former case, the A_{total} metric gives us an estimate of the overall project activity, which we can then use to rank projects according to.

Each asset type sum is decomposed to an aggregation of measurable entities weighted individually. As an example, equation 3 presents an expanded version of the A_{rep} aggregation. We denote A_i the total number of actions for action i. We also assign a weight w_i to each action type. Depending on whether an action has a negative or a positive contribution to the project, its value is added or retracted from the total score. Action aggregates for the remainder of the assets are expanded similarly.

$$A^{rep} = \sum_{i=0}^{n} w_i A_i = w_1 CAL \pm w_2 CNS \pm \ldots \pm w_{11} CBN \quad (3)$$

The most complicated aspect of our model is the calculation of the weights (w_i) for each action. For that, we use an approximative approach, exploiting the fact that the system this model is built for will extract information from many projects. Based on findings by Capiluppi et al. [2], we expect projects belonging to a single application domain to exhibit similar action patterns. For example, code development tools projects usually receive less documentation commits and feature more active mailing lists than educational projects. For this reason, we first create clusters of similar projects and then for each application cluster, we extract the action weights using the following algorithm:

1. Count all events for each actionID across all cluster projects

2. Count all events for all cluster projects

---

[1]By "lines of code" we refer to non-comment program statements.

Asset	Action	ID	Type
Code and	Add lines of code of good/bad quality	CAL	P/N
Documentation	Commit new source file or directory	CNS	P
Repository	Commit code that generates/closes a bug	CCB	N/P
	Add/Change code documentation	CAD	P
	Commit fixes to code style	CSF	P
	Commit more than X files in a single commit	CMF	N
	Commit documentation files	CDF	P
	Commit translation files	CDF	P
	Commit binary files	CBF	N
	Commit with empty commit comment	CEC	N
	Commit comment that awards a pointy hat	CPH	P
	Commit comment that includes a bug report num	CBN	P
Mailing lists -	First reply to thread	MRT	P
Forums	Start a new thread	MST	P
	Participate in a flamewar	MFW	N
	Close a lingering thread	MCT	P
Bug Database	Close a bug	BCL	P
	Report a confirmed/invalid bug	BRP	P/N
	Close a bug that is then reopened	BCR	N
	Comment on a bug report	BCR	P
Wiki	Start a new wiki page	WSP	P
	Update a wiki page	WUP	P
	Link a wiki page from documentation/mail file	WLP	P
IRC	Frequent participation to IRC	IFP	P
	Prompt replies to directed questions	IRQ	P

Table 1: Project resources and actions that can be performed on them. The Type column denotes whether an action has positive (P) or negative (N) impact.

3. The percentage of contribution of each action category to the total number of actions is the weight w_i for each action

We expect large projects to have thousands of events in each action type and therefore the values that we will be able to obtain through our heuristic method will be close to those that we would have obtained from a linear regression analysis of those weights, should that have been possible. An evaluation of a large number of projects for each cluster will provide us with indicative information about which actions are most prominent in software development and will allow us to filter out those actions that do not have a significant effect on developer contribution.

Finally, the contribution factor $CF(d)$ is a specialization of equations 2 and 3 per developer. To calculate the contribution factor value, we count the number of events for each event type and for each asset type that were performed by a specific developed d. If $A_i^a(d)$ is a function that returns the number of actions of type i performed by developer d on asset a and w_i^a is the weight of action i on asset a, then the contribution factor for the developer is calculated as show in equation 4.

$$CF(d) = \alpha \sum_{i=0}^{n} w_i^{rep} A_i^{rep}(d) + \cdots + \epsilon \sum_{i=0}^{n} w_i^{irc} A_i^{irc}(d) \quad (4)$$

4. IMPLEMENTATION

The model presented is being developed as an extension of the SQO-OSS system [5], and more specifically, as a plug-in to the Alitheia software evaluation tool. The Alitheia platform is an OSGi-based tool, targeted to the evaluation of software quality. It consists of a set of core services, such as accessors to project assets, continuous updating of monitored projects and relational data storage, and it is extensible through the use of plug-ins. Plug-ins can either implement basic software metrics or combine the results from other plug-ins arbitrarily. In fact, the contribution metric implementation is a compound plug-in, building on existing plug-ins that calculate basic metrics such as lines of code and mailing list participation. The system is designed to perform in-depth analysis of thousands of projects on a per repository revision basis and allows full automation of the quality evaluation process after the initial project registration.

5. EVALUATION AND CALIBRATION

To evaluate the validity of the proposed metric, we applied the Kaner and Bond metric evaluation framework [8]. The results can be seen in Table 2. The proposed metric captures well the scaling of the measured attribute. A careful selection of the metric components, i.e. the actions to be measured, will strengthen the metric's ability to produce accurate measurements.

To this end, we will combine questionnaires with statistical analysis. Questionnaires will solicit the views of developers (seen also as experts) regarding project assets and actions that provide contributions. Developers will be asked:

- to scale the importance of each action for a project

- to answer how these actions can be categorized into types

- to scale what the actions' perceived compatibility is with the given project assets

As a side-effect, respondents will also shed light on missing or redundant actions. Finally, we will use correlation analysis to identify and filter out actions that exhibit strong correlation coefficient.

Criterion	Our Metric
Purpose	Assess developer contribution in agile and distributed working environments.
Scope	A project developed by a distributed workgroup
Measured Attribute	Degree of contribution to the development process
Attribute Scale	Ratio scale
Attribute Variability	There is no knowledge of the variability of the measured attribute prior to performing the measurements
Metric Function	The proposed metric *counts* and *weights* the number of actions on project assets and the lines of code on a per developer basis. The highest those counts are, the more a developer has contributed to a project (see section 3)
Metric Scale	Ratio scale: The higher the contribution value, the more a developer has offered to the project.
Variability of readings	Some metric components are based on heuristics which may not work in certain cases. This may affect measurements in non-foreseeable ways. Metric components showing unstable results should be identified and excluded from the final version of the model.
Attribute and Metric Relationship	The metric generally captures changes in the attribute well. Metric components are analogous to contribution, subject to variability. For 2 given developers $d1$ and $d2$, the equation $c(d1) + c(d2) = c(d1 + d2)$ is always valid.
Side effects	No side effects can be foreseen. As the metric takes into account a variety of factors and it is automatically calculated it is difficult for developers to change their behavior towards optimizing the metric without increasing their actual contribution.

Table 2: Metric evaluation according to the Kaner and Bond framework

6. CONCLUSIONS

We presented a approach for evaluating developer contribution to the software development process, based on data acquired from software repositories and collaboration infrastructures. Our model introduces a set of predefined actions and exploits the availability of a large number of evaluated projects in the Alitheia tool to extract a weight of the importance of each action. As of this writing, we are implementing a plugin for a quality evaluation tool that produces measurements for each predefined action, while also recalculating the weights on regular intervals.

7. ACKNOWLEDGEMENTS

8. REFERENCES

[1] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *The 8th international workshop on economics-driven software engineering research*. ACM, May 2006.

[2] A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. In *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, pages 317–327, Mar 2003.

[3] D. N. Card and B. Scalzo. Measurement for object-orienated software projects. In *Proceedings of the 6th International Symposium on Software Metrics*, Florida, Nov 1999.

[4] R. L. Glass, I. Vessey, and S. A. Conger. Software tasks: intellectual or clerical? *Inf. Manage.*, 23(4):183–191, 1992.

[5] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, and D. Spinellis. Software quality assesment of open source software. In *Proceedings of the 11th Panhellenic Conference on Informatics*, May 2007.

[6] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32(7):1159–1177, Jul 2003.

[7] S. H. Kan. *Metrics and Models in Software Quality Engineering*, chapter 12.3 Productivity Metrics. Addison-Wesley, 2003.

[8] C. Kaner and W. Bond. Software engineering metrics: What do they measure and how do we know? In *10th International Software Metrics Symposium (METRICS 2004)*. IEEE, IEEE CS Press, Sep 2004.

[9] S. Koch and G. Schneider. Results from software engineering research into open source development projects using public data. Diskussionspapiere zum tätigkeitsfeld informationsverarbeitung und informationswirtschaft, Wirtschaftsuniversität Wien, 2000.

[10] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 263–272, New York, NY, USA, 2000. ACM.

[11] H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Commun. ACM*, 11(1):3–11, 1968.

[12] D. Spinellis. Global software development in the FreeBSD project. In P. Kruchten, Y. Hsieh, E. MacGregor, D. Moitra, and W. Strigel, editors, *International Workshop on Global Software Development for the Practitioner*, pages 73–79. ACM Press, May 2006.

[13] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, Jul 2003.