Measuring Developer Contribution from Software Repository Data

Eirini Kalliamvakou, Georgios Gousios, Diomidis Spinellis, Nancy Pouloudi
Department of Management Science and Technology
Athens University of Economics and Business
Athens, Greece
ikaliam,gousiosg,dds,pouloudi@aueb.gr

Abstract

*Our work is concerned with an enriched perspective of what constitutes developer contribution in software infrastructures supporting incremental development and distributed software projects. We use the term "contribution" to express the combination of all the actions a developer has performed during the development process and propose a model for calculating this individually for developers participating in a software project. Our approach departs from the traditional practice of only measuring the contribution to the final outcome (the code) and puts emphasis additionally on other activities that do not directly affect the product itself but are essential to the development process. We use the Open Source Software (*OSS*) context to take advantage of the public availability of data in software repositories. In this paper, we present our method of calculation and its system implementation and we apply our measurements on various projects from the* GNOME *ecosystem.*

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 monitor the rate of project development, identify implementation bottlenecks and isolate exceptional cases, while the results of contribution assessments can help with project planning and future estimations. An open issue linked 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 (LOC) or function points towards the final development of a software project [Kan, 2003]. This practice clearly focuses on the contribution to the final outcome of the project (i.e. the source code). To this end, only LOC is regarded as measured contribution. In recent years, however, the shift towards modern development practices and the proliferation of software and project management tools challenge this perspective. A software developer today is not only required to write code, but also to communicate and coordinate with colleagues effectively and to use a variety of tools that produce and modify code with minimal input from his or her side. This change has become more apparent with the emergence of Open Source Software (OSS).

In this respect, a developer contributes to a wide range of activities both involving the process and the product. Such an enriched perspective on a developer's contribution requires all individual actions to be taken into account. In this paper we discuss and measure contribution in this respect; a combination of all the actions a person has performed during the software development process weighted for their significance to the specific project. Our practice, then, encompasses the contribution to the final outcome as well as to the process that generated it.

This paper introduces a new model for measuring developer contribution, assuming that a more comprehensive image can be formed about a developer's contribution by combining actions directed towards the product itself and the process that yields it. For implementing our contribution calculation algorithm we have combined our proposed model of calculation with repository mining techniques. We provide a visual representation of the results, thus offering rich information regarding the total contribution per developer and how it is divided among different actions during the development process. Our initial observations set the basis for discussing contribution to multiagent, distributed software projects based on this new kind of information.

2. Existing work

We use the term "contribution" to express the combination of all the actions a developer has performed during the development process. In today's changing software development environment a developer's work items have been enriched with the addition of further activities that benefit the whole project, and this reality needs to be reflected. Contribution, as a notion, encapsulates other notions that have been frequently used in the literature to express activity, participation, effort or performance. In these cases, we see that although the name changes, the same concept is being described and the same measurement is used.

Productivity is a reoccurring discussion in all processes that involve inputs and outputs. In economic terms, productivity is the ratio of output to input, the output of a process divided by the effort required to produce it. In [Walston and Felix, 1977], programmer productivity is defined as the ratio of the delivered source lines of code (DSL) to the total effort in man-months (MM) required to produce the delivered program. Input and output in software engineering processes are frequently addressed with output usually measured in LOC [Walston and Felix, 1977], [Asundi, 2005], [Maxwell and Forselius, 2000]. As the LOC metric cannot be determined safely before the end of the project, function point analysis usually complements it. Input, on the other hand, is not as a straightforward notion in software development and its calculation requires further explanation.

In a software project there are several assets that receive input [Hertel et al., 2003], [Koch and Schneider, 2000], leaving trails of the actions of participating developers. Participation and performance of developers, which can be calculated from their input, are frequently discussed in productivity contexts. Again, although it is noted that OSS developers provide many different kinds of services to their projects, participation is measured in terms of number of source code contributions, showing a complete focus on participation to the outcome, while performance is mainly expressed in terms of rank advancement [Roberts et al., 2006].

The shortcomings of just measuring LOC to account for a developer's significance to a project have been discussed by researchers [Amor et al., 2006]. Aiming to estimate cost in the OSS context, Amor et al. propose that developer activity should be calculated. Although this is usually done by means of LOC, they stress that it is necessary to enhance this by a more detailed description of activity that accounts for actions other than simply writing code. This is a first attempt to move from focus on the outcome to examining the whole process. Cost is considered a function of effort, which in turn is considered a function of activity and suggested sources of information include CVS repositories, mailing list archives and bug tracking systems. In this regard, Amor et al. differentiate from previous literature that regards participation of developers simply as the addition of LOC [Koch and Schneider, 2002], [Mockus et al., 2002], [Mockus and German, 2003].

Today, with software development following more agile practices, developers in a project contribute to more project assets than simply writing code. Agile software development shares similarities with the OSS environment [Warsta and Abrahamsson, 2003] and here developers, too, have a multifaceted presence and contribution to the project, not only at the level of the code artifact but also in more supporting activities. Especially OSS projects lend themselves well to discussions and calculations of contribution due to the wide variety of publicly available data. To this end, we propose a definition and measurement of developers' contribution that accounts not only for the LOC that they have produced but also their support via posting to mailing lists, submitting bug reports and building wikis.

3. Our approach

Our work is concerned with the measurement of developer involvement and activity in the face of incremental and distributed development practices. The model we are building exploits the availability of publicly accessible software repositories to perform measurements and its system implementation can run fully automatically with no human intervention. The current paper extends previous work [Gousios et al., 2008], both theoretically as well as technically. Specifically, we present an updated method of calculation and a more detailed table of actions. Also, we have applied our methods and measurements to generate results.

Our model departs from the classic measurement practices as it does not consider the added lines of source code as the only contribution metric. This is a deliberate choice that we believe better reflects how software is developed using modern development methodologies, in the context of which, an important portion of development time is spent on communication and manipulation of development support tools. Our model does not neglect the importance of source code either; we still use the lines of code which we have represented via three actions (CADD, CREM, CCGN), but we also combine them with the developers' other actions on the project. We argue that this combination provides a more complete image of how much a developer has contributed to the software development process, not accounting only for writing code.

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 have been initially identified intuitively and through personal experience and based on related literature [Hertel et al., 2003], [Koch and Schneider, 2000], [Amor et al., 2006]. After consulting with experts the table is updated and refined.

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 [Spinellis, 2006]. Each action is a measurable entity whose value is updated after the corresponding project asset has been updated.

Not all actions have a positive effect on a project; for example, a commit with an empty commit comment can be considered as negative contribution. Furthermore, not all actions have the same importance on the evolution of a

Table 1. Project resources and actions that can be performed on them. The Effect column denotes whether an action has positive or negative impact.

Asset	Action	Id	Effect
Code and Documentation Repository	Add lines of code	CADD	$+$
	Remove lines of code	CREM	$+$
	Change lines of code	CCGN	$+$
	Commit new source file	CNS	$+$
	Commit new directory	CND	$+$
	Commit code that generates a bug	CGB	$-$
	Commit code that closes a bug	CCB	$+$
	Add/Change code documentation	CAD	$+$
	Commit fixes to code style	CSF	$+$
	Commit more than X files in a single commit	CMF	$-$
	Commit documentation files	CDF	$+$
	Commit translation files	CTF	$+$
	Commit binary files	CBF	$-$
	Commit with empty commit comment	CEC	$-$
	Commit comment that awards a pointy hat	CPH	$+$
	Commit comment that includes a bug report num	CBN	$+$
Mailing lists - Forums	First reply to thread	MFR	$+$
	Start a new thread	MST	$+$
	Participate in a flamewar	MFW	$-$
	Close a lingering thread	MCT	$+$
Bug Database	Close a bug	BCL	$+$
	Report a bug	BRP	$+$
	Close a bug that is then reopened	BCR	$-$
	Comment on a bug report	BCC	$+$
Wiki	Start a new wiki page	WSP	$+$
	Update a wiki page	WUP	$+$
	Link a wiki page from documentation/mail file	WLP	$+$
IRC	Frequent participation to IRC	IFP	$+$
	Prompt replies to directed questions	IRQ	$+$

project; for this reason, we also specify weights that are attached to each action.

We consider a project with a set of k developers (which we shall call $Developers$ throughout). Each one of them can perform any of the n different actions to contribute to a project. With each action i, we associate two functions, $c_i : Developers \rightarrow \mathbb{R}$ and $C_i : Developers \rightarrow [0,1]$. $c_i(d)$ represents the total number of actions identified for developer d with regard to action i, while $C_i(d)$ is the corresponding percentage, i.e., $c_i(d)$ divided by the sum of the work that all developers did in this action:

$$C_i(d) = \frac{c_i(d)}{\sum\limits_{j=1}^{k} c_i(d_j)}.$$

Since not all actions have a positive effect on the project, we can group actions together and derive separate calculations for positive-effect and negative-effect contributions of developers.

Furthermore, not all actions that constitute contribution to the project have the same importance. For this reason, the model also allows for weights to be attached to each action. These weights will be specified independently of the model, in order to reflect individual views regarding each action's significance to the whole project, for every different project.

We either use weights $w_1, \dots, w_n \in [0,1]$ with $\sum\limits_{i=1}^{n} w_i = 1$, or we may use arbitrary weights $W_1, \dots, W_n \in \mathbb{R}$ to represent significance. If we use the weights w_i, we compute the total contribution of each developer d by

$$C_{\text{tot}}(d) = \sum_{i=1}^{n} w_i C_i(d),$$

while in the case that we use W_i, we compute the weighted average:

$$C_{\text{tot}}(d) = \frac{\sum\limits_{i=1}^{n} W_i C_i(d)}{\sum\limits_{i=1}^{n} W_i}.$$

The model's invariant is that for any i, $\sum\limits_{j=1}^{k} C_i(d_j) = 1$.

4. Model evaluation

In order to evaluate our proposed metric, we have applied the Kaner and Bond metric evaluation frame-

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

Criterion	Our Metric
Purpose	Assess developer contribution in 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. The highest those counts are, the more a developer has contributed to a project in a positive or negative manner (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 in the same project, $d1$ 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.

work [Kaner and Bond, 2004]. Kaner and Bond propose their framework to evaluate software metrics through the measurement of which quality attributes can be captured and described. The framework denotes that the metric should possess certain properties in order to ensure that it fits the purpose of describing the quality attribute.

Currently we use the proposed contribution metric in its own merit but we see that it can be used also to explain causal relationships involving contribution since it captures well the scaling of the measured attribute. We use this as an evaluation of our metric for the purposes of this paper. The results can be seen in Table 2. In Section 7 we discuss how we plan to verify our method of calculation.

5. Implementation and experiment methodology

The model presented has been developed as a plug-in to the Alitheia Core software evaluation tool. The Alitheia platform is an extensible, open platform for software engineering research [Gousios and Spinellis, 2009]. Alitheia Core consists of a set of 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 various project data sources or from other plug-ins arbitrarily. Alitheia Core stores plug-in results differentialy, by attaching them to entities exported by its database. 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. We used the Alitheia Core tool to preprocess the

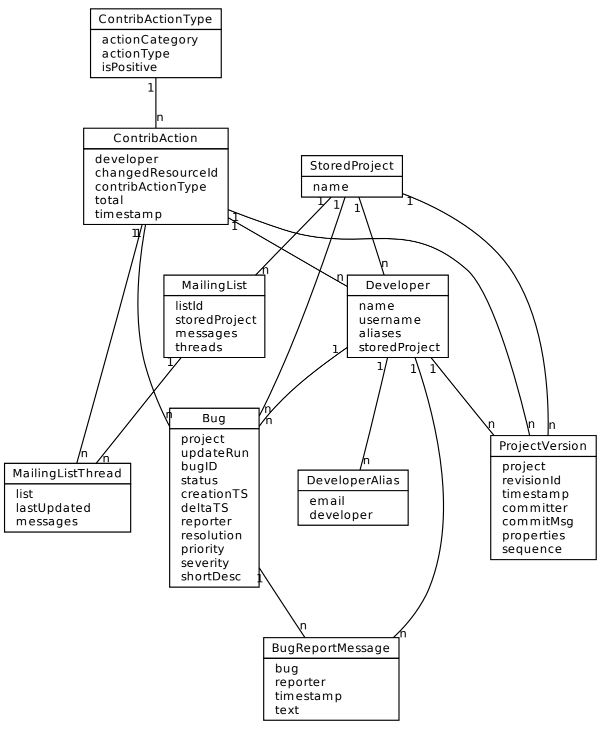

Figure 1. Relationships between entities defined by the Alitheia Core storage schema and those defined by the contribution plug-in

full history of the source code repositories, the full mailing list archives up to January 2009 and 3 years worth of bug reports from 48 sub-projects of the GNOME project.

The contribution plug-in is implemented as a compound plug-in, building on the pre-existing size metrics plug-in to avoid re-implementing them. The contribution plug-in is bound to three project entities, namely project versions, mailing list threads and bug reports. This means that it is automatically recalculated everytime the core system encounters an updated version of either of the three entities. The overall implementation is relatively straight forward: the plug-in makes extensive use of platform services, for example to recognize file types or to get threaded messages in order of arrival, in order to analyze the actions that the developer has performed on the affected resources.

The plug-in uses a custom table to extend the Alitheia Core default schema in order to store its results. The storage schema extention can be seen in Figure 1. For each identfied action, the plug-in stores the affected resource identifier, the developer identifier and also copies the timestamp of the affected resource.

A crucial point of the implementation is the identification of developer identities across the three data sources. In the course of a project, developers use several emails to post to mailing lists or to subscribe to bug tracking systems, but usually can be uniquely identified by the name that is attached to an email post or the user name for the project's SCM system. During the project updating phase, Alitheia Core fills the `Developer` table in with all data each updater knows or can infer from the raw data, namely user names, {`real name, email`} tuples and emails for source code, mailing lists and bug databases respectively. It then applies a set of heuristics, such as various anagrams of the developer's name and approximate string matching algorithms, to map developer names to SCM usernames. Identity resolution is currently not very effective: out of the 6137 unique usernames the system recognised for the projects we evaluated, only 598 were fully resolved. For this reason, we conducted all measurements on the set of identities that have been matched only. We performed manual inspection on a random set of matched identities to ensure the validity of the matching.

6. Results and discussion

We have performed our measurements on 48 sub-projects of the GNOME project. We have gathered data for 17 actions, 14 with positive effect and 3 with negative. Our data cover the whole history of the project until January 2009 for source code and mailing list-related actions, while we also processed the bug reports for the last 3 years.

In Figures 2 and 3 we present the visual representation of our results. This is a new type of information offered that can be used for discussion of various aspects of contribution, especially if combined with project-specific characteristics. We have chosen to present 4 projects (GNOME Desktop, GNOME-VFS, gedit and Tracker), where the percentage of resolved developers was greatest. For each project we can see how the total contribution of each developer is distributed among actions, accounting separately for positive-effect (up) and negative-effect (down) contribution. In these diagrams the information relates specifically to resolved developers.

Although our model supports action weights, for the purposes of this paper we have made all our calculations using equal weights of 1 for each action. As a result of this decision on the one hand we lose information regarding contribution in terms of significance to the project but, on the other, we see more clearly how developers decide to spread their contribution across different actions.

The view of contribution offered by these diagrams enables us to make a series of observations. Firstly, we can use them to focus on exceptional cases in a project and see what pattern the specific developers portray. For example we can see that in GNOME Desktop the developer with the highest contribution in positive-effect actions has a very low contribution in negative-effect actions. On the other hand, in GNOME-VFS we observe that the highest ranking developers in terms of positive-effect contribution also have the highest negative-effect contribution. Such observations might lead to different conclusions for each project, taking into consideration its specific characteristics. For example, a single developer in the GNOME-Desktop project has a high score of binary file commits; judging from the fact that the GNOME-Desktop project develops the user visible parts of the desktop, a possible explanation could be that the specific developer is a project artist that commits a large number of image files. Also, in the GNOME-VFS project, we observe that the developers who have done the most work, seem to also have performed the largest share of big number of files commits. This might be due the fact that the people that do the most work are project leaders and therefore are those that create branches of tags, which in turn makes them appear to have commited the most files.

We can use this type of results also to discuss the nature of the distribution of work carried out by developers. An important observation is that, in this initial stage, there doesn't seem to be an exclusive predominance of one action. Developers spread their contribution among several actions relating to all aspects, not showing a high degree of specialization.

More specifically, it is interesting to see that the three actions relating to the traditional LOC (CADD, CREM, CCGN), are not as dominant as would be expected. Indeed we see that not all participating developers contribute to these actions and that developers that do, also devote a substantial portion of their work in other actions, too. This supports our argument that strictly measuring code only speaks for a fraction of a developer's contribution and that this information needs

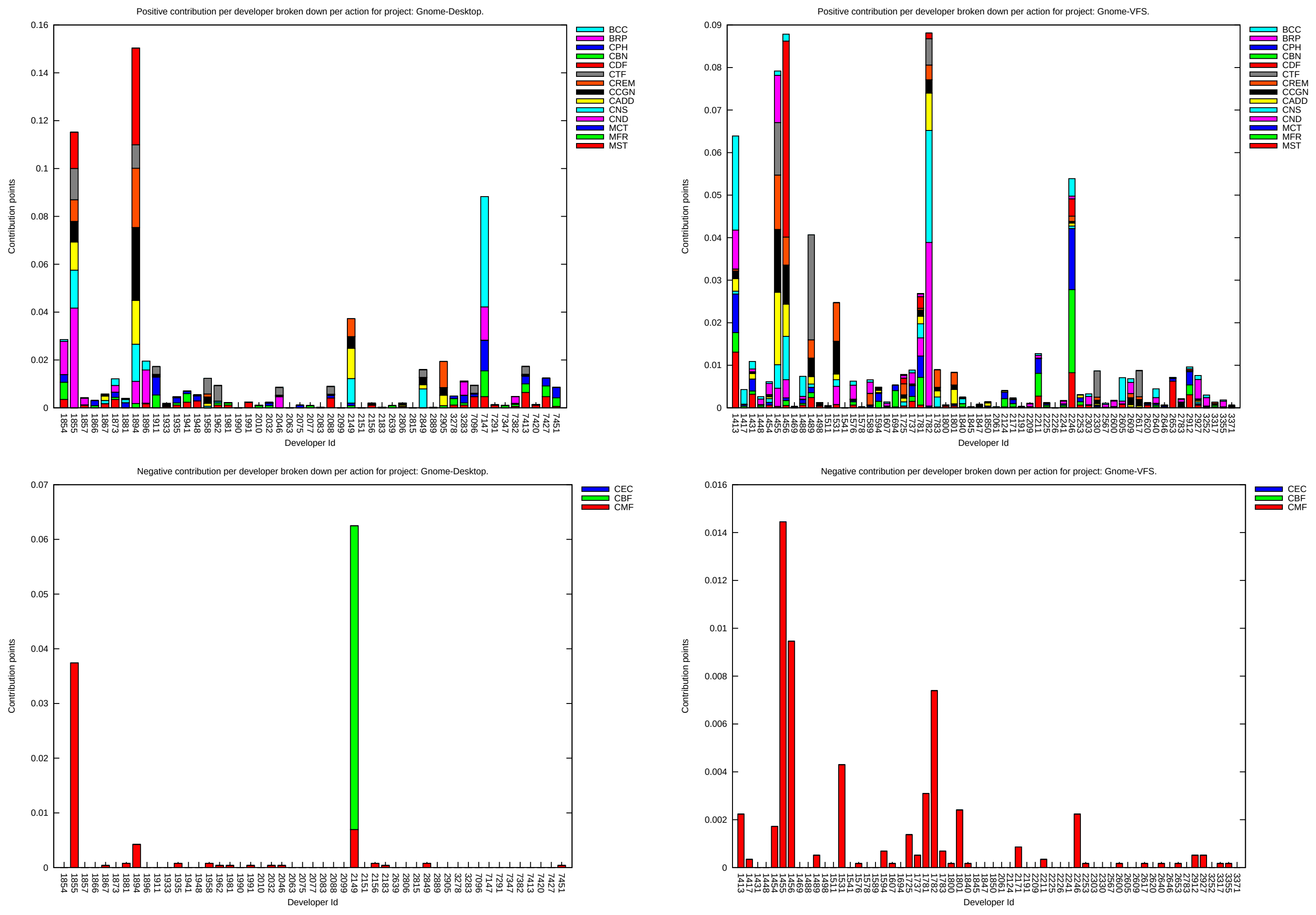

Figure 2. Positive (up) and negative (down) action distribution for various project developers

to be combined with activity in other domains of the process.

We have also used our data to check whether a Pareto-like principle applies to the set of projects that we have reviewed so far. For this purpose, we have prepared a diagram (Figure 4) that shows the total contribution percentage of the highest-ranking 30% of developers in each project.

The Pareto principle states that for many events, roughly 80% of the effects come from 20% of the causes, and has been found to apply to many software engineering processes [Boehm, 1987] and artifacts [Louridas et al., 2008]. Used for large sets of participants this can take the form of various combinations (60-40, 70-30 e.t.c). In our case we can see that for the set of all developers (both resolved and not), on average 70% of contribution comes from 30% of developers.

7. Limitations and further research

One limitation of our research relates to possible validity threats of the discussed methods. Firstly, our individual methods of calculating activities are not the only ones available. We have reviewed alternatives and have chosen those that are closer to our data types and organization. Although these may not be considered optimum, they are commonly applied to all projects and all involved developers, thus rendering no consistency problems.

Secondly, we have used in our data sets only those developers that we have successfully matched to all assets. Currently, there is no process that leads to more accurate results than manual matching. Due to the lack of an automated process and since manual matching is unlikely for such large numbers, we have relied on heuristics. Our methods provide satisfactory results, in some cases even better that previous methods, but still the developer sets we obtained are only 10% of the actual developers. We are currently investigating automated methods that will improve the ratio of matching to total developers so that we don't lose significant amounts of information.

There is an additional consideration regarding matching developers. We have assumed that only developers that are matched across all assets should be retained as valid

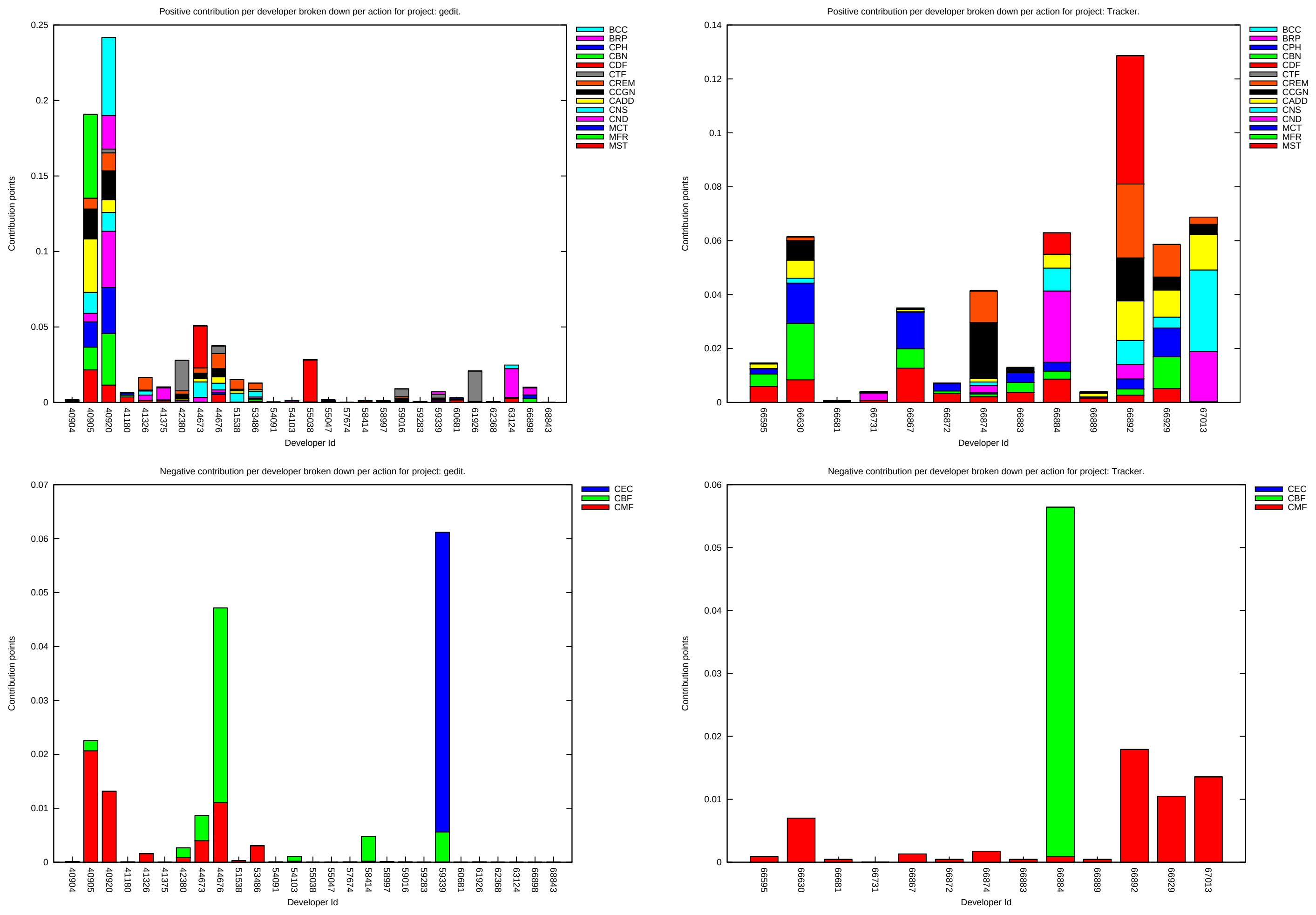

Figure 3. Positive (up) and negative (down) action distribution for various project developers

data. This assumption poses the threat that developers that are indeed active in only one aspect of the development process will be disregarded. In our view, however, such an assumption will not heavily distort the data as developers that are exclusively involved in only one asset are not common.

A final validity threat is posed by the fact that people have the ability to tailor their behavior to things they are measured against. Hence, some distortion is possible [Austin, 1996], [Weinberg and Schulman, 1974].

As it was mentioned earlier, we plan to further verify our method of calculation. For this purpose we will address our model as a formative model of measurement [Diamantopoulos and Winklhofer, 2001] and test it through a Partial Least Squares (PLS) model testing.

An additional goal regarding this research is to develop suitable techniques in order to mine data for additional actions and a larger number of projects, thus broadening our view of developer contribution with enriched information. The results offered by our proposed method and tool can be used to analyze and discuss patterns of developer contribution in a variety of contexts. An interesting aspect would be to incorporate the element of time and discuss how developer contribution levels change at different time intervals or between project milestones (e. g. releases).

8. Conclusion

In this paper, we have presented our work concerning the calculation of individual developer contribution to the software development process. We have formed a method for measuring contribution that encompasses actions of participation to the source code repository, the mailing lists and the bug tracking systems of software projects and applied this initially to several projects of the GNOME ecosystem. The resulting information, here demonstrated for a selection of projects, can be used to better our understanding regarding the nature of the distribution of work done by developers and enhance the research agenda in OSS. Future research activities include the use of this information on a larger

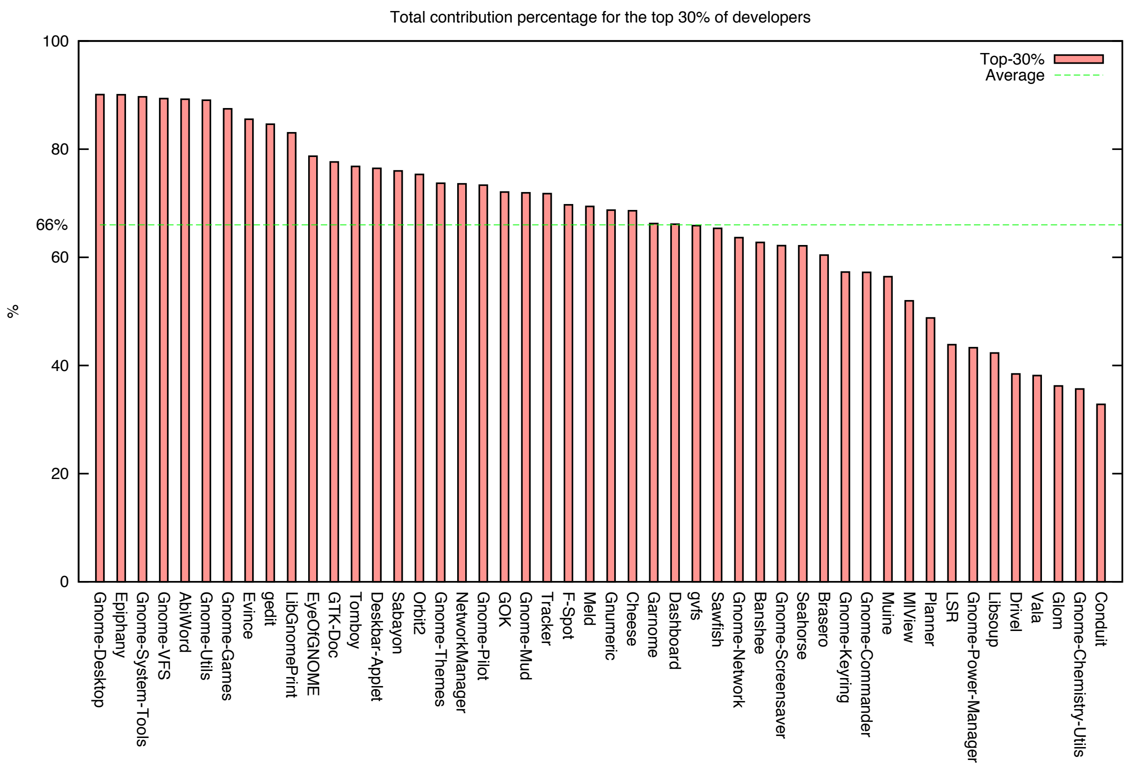

Figure 4. Total contribution from the top 30% of developers for various projects.

scale of projects and its combination with additional data for clusters of projects for performing analyses.

The full source code for the Alitheia Core and contribution metric plug-in can be found online at http://www.sqo-oss.org.

Acknowledgements

This work was partially funded by the European Community's Sixth Framework Programme under the contract IST-2005-033331 "Software Quality Observatory for Open Source Software (SQO-OSS)". Project contributors include the Aristotle University of Thessaloniki, Prosyst Software GmbH, Sirius plc, Klarälvdalens Datakonsult AB and members from the KDE project community. The authors would like to thank Stavros Grigorakakis for his help in organising the mirrored project data.

References

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

[Asundi, 2005] Asundi, J. (2005). The need for effort estimation models for open source software projects. In *5-WOSSE: Proceedings of the fifth workshop on Open source software engineering*, pages 1–3, New York, NY, USA. ACM.

[Austin, 1996] Austin, R. D. (1996). *Measuring and Managing Performance in Organizations*. Dorset House Publishing Company, Incorporated.

[Boehm, 1987] Boehm, B. W. (1987). Industrial software metrics top 10 list. *IEEE Software*, 4(9):84–85.

[Diamantopoulos and Winklhofer, 2001] Diamantopoulos, A. and Winklhofer, H. M. (2001). Index construction with formative indicators: An alternative to scale development. *Journal of Market Research*, 38(2):269–277.

[Gousios et al., 2008] Gousios, G., Kalliamvakou, E., and Spinellis, D. (2008). Measuring developer contribution from software repository data. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 129–132, New York, NY, USA. ACM.

[Gousios and Spinellis, 2009] Gousios, G. and Spinellis, D. (2009). Alitheia core: An extensible software quality monitoring platform. In *Proceedings of the 31rst International Conference of Software Engineering - Research Demos Track*, Vancouver, CA. IEEE. To appear.

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

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

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

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

[Koch and Schneider, 2002] Koch, S. and Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27–42.

[Louridas et al., 2008] Louridas, P., Spinellis, D., and Vlachos, V. (2008). Power laws in software. *ACM Transactions on Software Engineering and Methodology*, 18(1):1–26. Article 2.

[Maxwell and Forselius, 2000] Maxwell, K. D. and Forselius, P. (2000). Benchmarking software-development productivity. *IEEE Softw.*, 17(1):80–88.

[Mockus et al., 2002] Mockus, A., Fielding, R., and Herbsleb, J. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346.

[Mockus and German, 2003] Mockus, A. and German, D. (2003). Automating the measurement of open source projects. In *Proceedings of the 25th Workshop on Open Source Software Engineering (ICSE '03), Portland, Oregon*.

[Roberts et al., 2006] Roberts, J. A., Hann, I.-H., and Slaughter, S. A. (2006). Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Manage. Sci.*, 52(7):984–999.

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

[Walston and Felix, 1977] Walston, C. E. and Felix, C. P. (1977). A method of programming measurement and estimation. *IBM Systems Journal*, 16(1):54–73.

[Warsta and Abrahamsson, 2003] Warsta, J. and Abrahamsson, P. (2003). Is open source software development essentially an agile method? In Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K., editors, *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 143–147. International Conference on Software Engineering.

[Weinberg and Schulman, 1974] Weinberg, G. and Schulman, E. (1974). Goals and performance in computer programming. *Human Factors*, 16(1):70–77.