Detecting Agility of Open Source Projects Through Developer Engagement

Paul J. Adams[1] and Andrea Capiluppi[2] and Adriaan de Groot[1]

[1] Sirius Corporaion Ltd., Hamm Moor Lane, Weybridge, UK, KT15 2SF
paul.adams@siriusit.co.uk, groot@kde.org
[2] Centre for Research on Open Source Software, University of Lincoln, Lincoln, UK, LN5 7TS acapiluppi@lincoln.ac.uk

Abstract. The principles behind the agile development methods and common practise within the Open Source community are vastly different. In recent years there has been a rise of interest in these, in order to detect and inform on areas of compatible shared practises. This paper argues that it is possible to quantify the level of agility displayed by Open Source projects. An indicator of agility, the Mean Developer Engagement (MDE) metric is introduced and tested through the analysis of public project data. Projects sampled from two repositories (KDE and SourceForge) are studied and a hypothesis is formulated: projects from the two samples display a similar level of MDE.
This paper provides two main contributions: first, the MDE metric is shown to vary significantly between the KDE and SourceForge projects. Second, by combining MDE with a project's lifespan, it is also shown that SourceForge projects have insufficient uptake of new developers resulting in more active, shorter, initial activity, and in a quicker "burning out" of the projects.

1 Introduction

Plan-driven approaches to software engineering emphasise large-scale planning and formal communications while agile approaches emphasise flexibility [14]. Proponents of agile processes argue that such processes respond to change and withstand the continual pressures of software evolution better than plan-driven approaches [6, 1]. Proponents of the OSS paradigm claim that it should be considered as a novel approach, and its evolution and maintenance phases should be considered different from a plan-driven approach [5].

In theory agility and Open Source are very different concepts; the latter being just a licensing paradigm with implications for code reuse and redistribution. Comparative studies have been made in the past, but the scarcity of data from Agile processes leaves most studies on the surface of theoretical discussions [19, 13].

The term "agility" has been formulated to assess how a software company locates itself in a spectrum between Agile and plan-driven. Process-related characteristics have been identified to help either in focusing on the critical factors of the development framework [4]; or in the selection itself of the most appropriate approach [7]. Agility in Open Source projects, however, is often implemented significantly differently from the canonical definition of the agile processes. Usually this occurs due to certain agile

practises requiring team co-location [15] or smaller teams [12]. Instead agility within Open Source is often performed by much larger teams in a distributed fashion; although co-location for certain activities, like sprinting, is becoming more common [9]. The unifying aspect of all agile processes is flexibility in order to react to change (in requirements, or other conditions) [3]. We posit that this, coupled with the agile culture of producing useful, working software as quickly as possible, creates an implicit requirement for an agile project to make efficient use of its developer resource.

This paper introduces and justifies the Mean Developer Engagement (MDE) metric. Being it a process-based metric, it correlates with the effort of OSS developers along the duration of a project, and it addresses one of the relevant factors in the definition of agility [4]. The metric is not introduced per-se: it is used both to empirically evaluate the agility of OSS projects, and to compare two samples of projects extracted from two repositories (KDE and SourceForge), in order to detect differences in the agility of OSS developers. The traditional Goal-Question-Metric paradigm will be used to detect the presence of differences: a research hypothesis will be formulated and a statistical test will be evaluated to detect statistically significant differences among the samples.

This paper is structured as follows: Section 2 introduces the MDE metric and describes the factors it is built upon, showing how the data was gathered and processed. Section 3 introduces the GQM approach and tailors it into the research hypothesis which this paper explores. Section 4 evaluates the results and provides the discussion of the statistical tests used. Section 5 concludes.

2 Mean Developer Engagement

Users and developers are limited resources within the OSS environment, particularly so with regards to develoeprs [8]. However, several major OSS projects have been noted to be particularly successful at recruiting new developers [16, 17]. No empirical evidence has been given yet to a comparison of how repositories employ their resources, in terms of active developers.

In order to draw a comparison between two OSS repositories, the Mean Developer Engagement (MDE) is here proposed and analysed as a metric. MDE is a measurement of how effective (on average over time) an OSS project is at making use of its human resources and, therefore, potentially indicative of agility [7]. MDE is measured over n time periods, and each developer associated with the project is classified as active or inactive in each period. Typically, a week is used as time period; the number n ranges from 1 (a very short project) to over 600 (long-term projects). In our research we define MDE as:

$$\bar{de}_n = \frac{(n-1).\bar{de}_{n-1} + \left(\frac{\text{dev}(active)}{\text{dev}(total)}\right)_i}{n} \tag{1}$$

– dev($active$) is the number of (distinct) developers active in time period i.

– dev(*total*) is the total number of developers involved with the project in the periods $0 \ldots i$.
– n is the number of time periods over which the project has been evaluated.

An ideal project, in which every developer is active in every single time period within the range of the MDE calculation, has an MDE of 1. The primary concern with the current definition of MDE lies with the metric dev(*total*), this metric naïvely asserts that the number of accounts in the project repository is the total number of developers in the project. This is deficient as an account may remain in the repository long after a developer has left a project. To create a refined and more precise definition, $dtotal$ is augmented with the inclusion of a *grace period*.

2.1 Refined Definition – Grace Period

OSS project developers leave a trace which can be retrieved by assessing version control logs (*i.e.* their account name appears for the first time). The definition of dev(*total*), above, establishes a list of everyone who has ever been a developer in a specific project. In order to detect when a developer leaves a project, it is possible to set an arbitrary time span after which a developer who has shown no activity in that span is removed from the list of project participants.

The definition of dev(*total*), for each n, should be refined in order to detect developer inactivity. Depending on length of service, each developer shall be allowed an inactivity "grace period," as are shown in Table 1 (These figures were created as a discussion exercise between 10 Open Source developers and validated by 10 more). Only once they have been inactive for longer than this time shall they no-longer be considered as project developers. As soon as the developer commits more code, they are reconsidered as a developer and their grace period is set to one, as a new developer.

Length Of Service	Grace Period
1024	20
520	15
416	12
260	10

Length Of Service	Grace Period
208	8
104	6
52	4
24	2

Table 1. Developer grace periods (in weeks), shorter length of service has grace period 1.

By making these adjustments a more accurate measure of dev(*total*) is calculated, thus improving the overall accuracy of MDE. The downside to this adjustment, however, is an increase in overall computational complexity when automating MDE measurement. This is simply a linear increase, with n, in complexity created by the adjust being made after each time quanta.

2.2 Data Gathering and Processing

In order to calculate MDE for a project we require to know which members of the project are active and when. In this research the data has been gathered from the history of the projects' source code repositories. This data could easily be supplemented with data from mailing lists, IRC logs, etc. The history contains, for each change done to the project, at least the user name of the developer and the date/time stamp of the commit. MDE can easily be calculated using these. Most source code management tools (such as Subversion) provide a way to print out the histories in a machine-parseable form.

Figure 2.2 shows example plots of MDE (in both simple and refined forms) for the entire KDE project (`http://www.kde.org`). KDE is a very large project, with over 1600 accounts in its Subversion repository, over 760000 changes and 10 years of development history. The plot shows the change in MDE as calculated on a week-by-week basis from the beginning of the project in 1997.

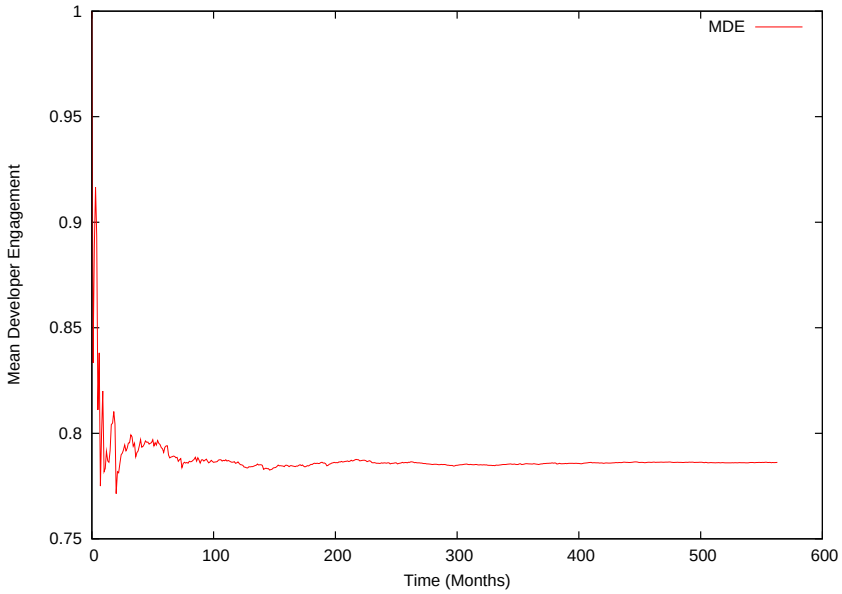

Fig. 1. MDE over the lifetime of the KDE project

The first aspect of MDE plots that should be noted is the anomalies which occurs within the data for the first year. These deviations from the trends are caused by a disproportionate shift in the number of active developers early on. Early in a project's

lifetime, both n and dev($total$) are low and any change in dev($active$) will have a dramatic affect on MDE.

The initial period of upheaval is followed by an upturn in MDE, indicating that the project has attracted many new developers. This is shortly followed by a slightly steeper gradient in the plot, indicating increased turnover (i.e. an increased number of developers having short tenure in the project). As the project matures, this plot begins to level out as more developers maintain longer commitments to KDE. From week 300 until the end of the plot the gradient of MDE is stable. This is indicative of process (explicit or otherwise) and a stable community.

By applying the adjustments described by Table 1 the new plot is always equal to or greater than the original plot at any point in time. This is clearly shown in Figure 2.2 where the refined MDE plot is significantly higher in value. In the case of this KDE plot, the most remarkable aspect is the stability and success with which KDE has managed to engage its active developer base; this plot clearly indicating that the average regular contributor has been active nearly 80% of the time since the second year of the project.

3 MDE – Empirical Evaluation

In order to validate the MDE metric, the following Section will both apply it and study its behaviour on two samples of OSS projects: one was extracted from KDE, and the other was randomly selected from the repositories hosted at SourceForge (`http://sourceforge.net`). SourceForge (SF) hosts over 120,000 different projects; the majority of these are small projects compared with a large and long-term project like KDE (which is outside SourceForge) [10].

The MDE metric applied to the KDE project as a whole is analysed in Figure 2.2 above. The KDE project has to be considered as a collection of sub-projects, each of them much more focused on a particular goal, and with a smaller developer base. Very few of the developers in KDE are active across the entire project, as most focus on their own sub-project. This improves the accuracy of MDE for each sub-project by disregarding developers who work elsewhere in KDE.

As the data-base for the present work, 20 sub-projects from KDE and 20 SF projects were randomly sampled. Projects which had no history, (i.e. "dead on arrival" as they have not ever engaged developers) were replaced by other randomly selected projects. A research hypothesis was formulated based on the MDE behaviour on these two samples. It is tested first via a bi-directional test, to detect major differences between the two samples; then a directional test is applied to detect whether one sample statistically achieves better results in terms of the MDE metric.

3.1 Empirical Approach – Goal-Question-Metric

Metric-based evaluation of software products is well established within the software engineering research, since it allows a simple and effective means of quality evaluation [11]. The Goal-Question-Metric (GQM) approach [2, 18] is particularly popular

due to its ease of automation. This research was conducted using the GQM approach, with MDE as a metric.

Goal: The goal of this research is to identify, through the application of metrics, the degree of agility displayed by an Open Source development project. Empirically comparing the Agile and OSS development approaches, finding similarities and differences, will help bridge the gap between the traditional closed-source and newer paradigms in software development.

Question: As part of a greater research programme this study is conducted to establish the statistical significance of MDE as a metric. As such, we ask: Are projects within KDE statistically different from those hosted within SourceForge with regards to MDE? Alternatively, does KDE display a different level of agility than SF projects? If a difference is found in the bi-directional comparison of the samples, a more restrictive test will be formulated, asking whether the KDE sample consistently achieves better MDE results than the SourceForge sample.

Metric: The metric applied is MDE using time quanta of one week. The metric is applied across the project lifetime and an average MDE score for each week is used.

3.2 The Samples

The complete version control log of the 20 randomly selected KDE projects and SourceForge projects were parsed to produce the MDE scores shown in Table 2. Here, the date of the first log entry, i (the duration of the log in weeks) and the MDE score are shown for each project. In addition a "effort" value is included and will be described in the following section.

4 Results and Analysis

From the SourceForge sample, one may note the relatively high incidence of projects with an MDE of 1.0, indicating "ideal" projects with total engagement of all the project developers. A closer examination of the table shows a strong correlation between an MDE of 1.0 and a lifespan of one week; a project *necessarily* has an MDE of 1.0 in its first week, as the determination of active and total developers is based on the set of developers seen in that first week. Such short-lived projects are likely "dead on arrival" as they fail to engage any long-term effort [10]. Much more interesting are the longer-lived projects with a high MDE, such as wxpropgrid (MDE=0.9968 over seven months) and shareaza (MDE=0.7830 over three years).

4.1 Testing of the Hypothesis and Discussion

The Wilcoxon two-sample test is used to assess the significance in similarity between samples from two populations. It is applied here principally in two-tail form to show that the samples are indeed significantly different and yields: $W = 97$, $p \leq 0.005376$. At the 95% confidence level, this is a significant result. We reject the null hypothesis that the populations display similar levels of agility. We test the hypothesis that SF

KDE					*SF.net*				
Project	Start	*i*	MDE	Effort	Project	Start	*i*	MDE	Effort
dolphin	21-11-06	54	0.6799	36.7146	askcms	29-06-06	1	1.0	1.0
k3b	26-03-01	349	0.5961	208.0389	awdotnet	24-05-07	1	1.0	1.0
katomic	29-06-99	438	0.3340	146.2920	dvdshop	31-03-06	1	1.0	1.0
kcalc	13-04-97	554	0.4307	238.6078	hivex	16-07-07	1	1.0	1.0
kgeography	07-03-04	38	0.5386	20.4668	interaction	04-03-07	1	1.0	1.0
kig	15-04-02	294	0.6878	202.2132	kuragari	13-01-07	1	1.0	1.0
kivio	02-12-00	365	0.5320	194.1800	kyrios	03-07-06	74	0.5634	41.6916
kmail	18-01-03	254	0.6730	170.9420	map	06-11-05	50	0.3780	18.9000
kmoon	27-09-98	478	0.2913	139.2414	neuralbattle	14-06-06	74	0.6803	50.3422
knotes	30-06-97	544	0.4638	252.3072	opulus	25-07-07	10	0.4931	4.9310
kolourpaint	10-10-03	214	0.6269	134.1566	pwytter	09-07-07	1	1.0	1.0
konqueror	09-02-99	459	0.6610	303.3990	pyaws	11-04-06	56	0.2514	14.0784
konsole	28-10-98	474	0.6109	281.5666	rejuce	02-08-06	10	0.5554	5.5540
kontact	18-01-03	254	0.5867	149.0218	rlcyber	02-07-06	17	0.7612	12.9404
kopete	02-01-02	308	0.7142	219.9736	shareaza	02-06-04	182	0.7830	142.5060
kscd	04-07-97	542	0.4962	268.9404	stellarium	12-07-02	280	0.6183	173.1240
kspread	18-04-98	502	0.6216	312.0432	tclshp	04-10-06	1	1.0	1.0
ksudoku	03-03-07	39	0.6530	25.4670	tsg	23-03-07	19	0.6036	11.4684
kteatime	16-04-99	450	0.3299	148.4550	wxpropgrid	16-04-07	31	0.9968	30.9008
marble	29-09-06	61	0.6321	38.5581	xml-copy-editor	16-08-07	14	0.7731	10.8234

Table 2. MDE Scores Calculated for 20 Random Projects from KDE and SF.net.

shows greater level of agility than KDE using a single-tail Wilcoxon test and find: $W = 97$, $p \leq 0.002688$. This is another strong result at the 95% confidence level, and we reject the null hypothesis. SF *does* show greater agility.

The Wilcoxon two-sample tests applied here strongly indicate a statistical difference between MDE in the samples from KDE and SF and potentially a difference in agility. We may say that the SF projects achieve better developer engagement in the sampled projects. It may be surprising that both null hypotheses were rejected, especially given SourceForge's notoriety for being home to low quality projects [10].

This counter-intuitive result was studied further: an analysis of the start and end dates in the projects from both samples shows that KDE projects may not be as strong at utilising developers, but typically have longer lifespans. There may be further significance to MDE in conjunction with the lifespan of a project. Using the same samples from KDE and SF, the duration of the project was multiplied by the MDE value in order to produce a new "effort" score. The weighted effort value for each project is shown within Table 2.

Even without formal statistical analysis, we see a clear separation between the new "weighted" scores for MDE. The same two-tail Wilcoxon test applied to effort returns: $W = 374$, $p \leq 2.455e\text{-}06$. Therefore, at the 95% confidence level we can state that a randomly selected KDE project will display better developer engagement over time than a randomly selected SourceForge project. This can be explained as follows: SF projects tend to better engage developers, but just for a limited amount of time, and its projects tend to "burn bright" but fade rather quickly. The KDE project manages in-

stead to achieve a prolonged engagement of its developers, and this by itself represents a quality factor within different OSS repositories.

5 Conclusions and Further Work

This paper argued that it is possible to measure the agility of OSS projects by means of a metric based on developers effort. The Mean Developer Engagement (MDE) metric was introduced as a means for assessing how affectively OSS projects utilise their developer resource, if they leverage it consistently and for prolonged periods of time, and whether different OSS repositories achieve different levels of engagement of developers. Samples from two OSS repositories (KDE and SourceForge) were extracted, and the MDE's statistical significance was evaluated through a bi-directional and a single directional tests.

The empirical evaluation of the hypothesis showed at first a counter-intuitive result: if fact, it was found that SF projects are statistically better at engaging their developers than the projects from KDE. Investigating the results further, the duration of development was studied, which displayed much longer development periods in the KDE sample than in the SF one. Combining the duration with the engagement, we find that SF projects tend to better engage their developers, but only for a limited amount of time, after which it is common for the project to quit its activity. KDE projects have more overall endurance: part of the endurance beyond the lifespan of a SF project must be attributed to the more regular intake of new developers these projects have. Therefore a project is more likely to sustain MDE in a forge of related projects, such as KDE, than in a forge of unrelated projects, such as SourceForge.

Before MDE can be formally introduced as a metric further work shall be carried out to compare the results of Open Source MDE, presented here, with known agile project data from industry. This will allow us to establish upper and lower thresholds for MDE as an indicator of agility.

6 Acknowledgement

This work is partially supported by the European Community Framework Programme 6, Information Society Technologies key action, contract number IST-033331 ("SQO-OSS").

References

1. A. Capiluppi and J. Fernandez-Ramil and J. Higman and H. C. Sharp and N. Smith. An Empirical Study of the Evolution of an Agile-Developed Software System. In *International Conf. on Software Engineering*, pages 511–518, Minneapolis, Minnesota, May 2007.
2. V. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, pages 528 – 532. John Wiley & Sons, Inc., 1994.

3. K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2004.
4. B. Boehm and R. Turner. Using Risk to Balance Agile and Plan-Driven Methods. *IEEE Computer*, 36(6):57–66, 2003.
5. A. Capiluppi, J. Gonzales-Barahona, I. Herraiz, and G. Robles. Adapting the "Staged Model for Software Evolution" to Free/Libre/Open Source Software. In *Proc. Int'l Workshop on Principles of Software Evolution (IWPSE)*, Dubrovnik, Croatia, 3-4 Sept. 2007.
6. N. Chapin. Agile Methods' Contributions in Software Evolution. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, page 522, Washington, DC, USA, 2004. IEEE Computer Society.
7. S. Datta. Agility Measurement Index: A Metric for the Crossroads of Software Development Methodologies. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 271–273, New York, NY, USA, 2006. ACM.
8. A. de Groot, S. Kügler, P. J. Adams, and G. Gousios. Call for Quality: Open Source Quality Observation. In E. Damiani and B. Fitzgerald and W. Scacchi and G. Scotto, editor, *IFIP International Federation for Information Processing: Open Source Systems*, pages 57 – 62, 2006.
9. B. Düring. *Extreme Programming and Agile Processes in Software Engineering*, chapter Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project. Springer, 2006.
10. R. English and C. M. Schweik. Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. In *International Workshop on Emerging Trends in FLOSS Research and Development*, 2007.
11. Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, UK, 1991.
12. K. Schwaber and M. Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2002.
13. S. Koch. Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion. In *Extreme Programming and Agile Processes in Software Engineering: Proceedings the 5th International Conference XP 2004*, number 3092 in Lecture Notes in Computer Science (LNCS), pages 85–93. Springer Verlag, 2004.
14. M. Fowler and J. Highsmith. The Agile Manifesto. In Software Development, Issue on Agile Methodologies, `http://www.sdmagazine.com`, last accessed on March 8th,2006, August 2001.
15. M. Kircher and P. Jain and A. Corsaro and D. L. Levine. *Extreme Programming Perspectives*, chapter Distributed Extreme Programming. Pearson Education, 2002.
16. A. Mockus, R. .T. Fielding, and J. .D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.
17. G. Robles and J. M. Gonzlez-Barahona. Contributor Turnover in Libre Software Projects. In Ernesto Damiani, Brian Fitzgerald, Walt Scacchi, Marco Scotto, and Giancarlo Succi, editors, *OSS*, volume 203 of *IFIP*, pages 273–286. Springer, 2006.
18. R. van Solingen. *The Goal/Question/Metric Method: A Practical Guide For Quality Improvement Of Software Development*. McGraw-Hill, 1999.
19. J. Warsta and P. Abrahamsson. Is Open Source Software Development Essentially an Agile Method? In *3rd Workshop on Open Source Software Engineering*, 2003.