Towards A Portfolio of FLOSS Project Success Measures

Kevin Crowston, Hala Annabi, James Howison and Chengetai Masango
Syracuse University School of Information Studies
{crowston, hpannabi, jhowison, cmasango}@syr.edu

Abstract

Project success is one of the most widely used dependent variables in information systems research. However, conventional measures of project success are difficult to apply to Free/Libre Open Source Software projects. In this paper, we present an analysis of four measures of success applied to SourceForge projects: number of members of the extended development community, project activity, bug fixing time and number of downloads. We argue that these four measures provide different insights into the collaboration and control mechanisms of the projects.

1. Introduction

We are interested in identifying factors that predict distributed team performance, specifically, the performance of Free/Libre Open Source Software development teams on FLOSS development. In this paper, we take a first step in this direction by developing measures for the success of FLOSS projects.

It is important to develop measures of success for FLOSS projects for at least two reasons. First, having such measures should be useful for FLOSS project managers in assessing their projects. In some cases, FLOSS projects are sponsored by third parties, so measures are useful for sponsors to understand the return on their investment. Second, FLOSS is an increasingly visible and copied mode of systems development. Millions of users depend on FLOSS systems such as Linux (and the Internet, which is heavily dependent on FLOSS tools), but as Scacchi [1] notes, "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success". A recent EU/NSF workshop on priorities for FLOSS research identified the need both for learning "from open source modes of organization and production that could perhaps be applied to other areas" and for "a concerted effort on open source in itself, for itself" [2]. But to be able to learn from teams that are working well, we need to have a definition of "working well".

2. Measuring project success

Crowston, Annabi and Howison [3] examined the process of FLOSS development to suggest measures indicative of success for these projects. They noted that conventional measures of project success [e.g., 4, 5] focus primarily on use and the use environment of an information system, but for FLOSS, the use environment is generally very difficult to observe, while by contrast the development environment is quite visible. Furthermore, conventional measures define success from the perspective of corporate users who purchase the software, while FLOSS projects depend on the continued motivation of volunteer developers. Crowston, Annabi and Howison [3] therefore suggested developing a portfolio of success measures that includes measures of the development process. While they suggested a number of possible measures, they did not actually present data for any projects.

In this paper, we take the first steps towards implementing a portfolio of success measures. Specifically, we analyze four possible success measures, namely number of members of the extended development community, project activity, bug-fixing performance and number of downloads. These measures were adopted from the list presented by Crowston, Annabi and Howison [3] because they span the development process, including inputs (number of developers), process (project activity and bug fixing) and output (number of downloads).

Our analysis aims at assessing the construct validity of these measures. Each has good face validity, in the sense that a project that attracts developers, maintains a high level of activity, fixes bugs and that many users download does seem like it deserves to be described as a success. However, we are also interested in assessing how these measures relate to one another: do they measure the same construct or are they measuring different aspects of a multidimensional success construct? And most importantly, what insight do they provide into the nature of the development processes in the different projects?

In the following sections of the paper, we will discuss the procedure we used to obtain data with which to operationalize the measures, followed by the details of the analysis approach. We then present the results of this analysis and discuss the implications of these results. We conclude with some suggestions for future research.

3. Data

In this section we discuss the sample of projects selected and the process we followed to gather data about these projects.

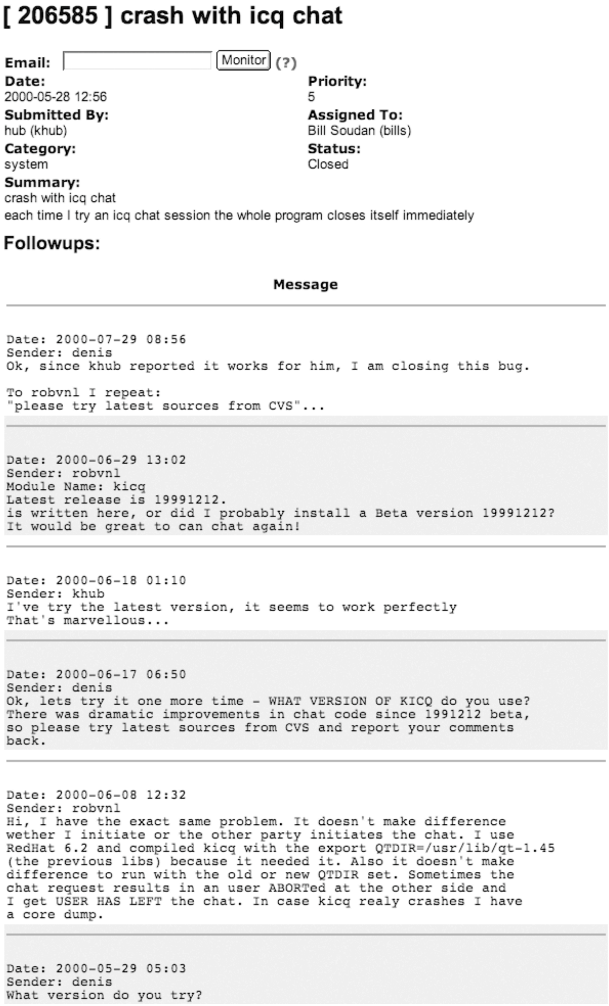
3.1. Project selection

To create a sample of FLOSS projects, we selected from projects hosted by SourceForge, a free Web-based system that provides a range of tools to facilitate FLOSS development (http://sourceforge.net/). At the time we started our study, SourceForge supported more than 50,000 FLOSS projects on a wide diversity of topics (as of 21 March 2004, the number was 78,003). Clearly not all of these projects were suitable for our study: many are inactive, previous studies have suggested that many are in fact individual projects [6], and some do not make bug reports available. Therefore, we restricted our study to projects that listed more than 7 developers and had more than 100 bugs in the project bug tracker at the time of selection in April 2002. Being listed as a developer grants write access to the project's code base, so projects with multiple developers are ones that might be expected to experience significant coordination issues. Having bug reports was a necessary prerequisite for the planned analysis, as well as indicative of a certain level of development effort. Quite surprisingly, we identified only 140 projects that met both criteria. Unfortunately, space does not permit a full listing of the projects, but the sample includes the projects *curl, fink, gaim, gimp-print, htdig, jedit, lesstif, netatalk, phpmyadmin, openrpg, squirrelmail* and *tcl*. Those familiar with FLOSS may recognize some of these projects, which span a wide range of topics and programming languages.

3.2. Data Collection

Two of the measures we included (activity level and number of downloads) are tracked by SourceForge and thus available directly from the project description page. As a measure of activity, we used the project activity rank. To gather these data, we developed a spider to download and parse the project pages. Data were collected in April 2003.

To study the performance of bug fixing, we collected data from the SourceForge bug tracking system, which enables users to report, and developers to discuss bugs. As shown in Figure 1, a bug report includes a description of a bug that can be followed up with a trail of correspondence. Basic data for each bug includes the date and time it was reported, the reporter, priority and, for closed bugs, the date and time it was closed. To collect this data, we developed a spider program that downloaded and parsed all bug report pages for the selected projects. Unfortunately, between selection of projects and data collection, some projects restricted access to bug reports, so we were able to collect data for only 122 projects.



[206585] crash with icq chat

Email: Monitor (?)

Date:
2000-05-28 12:56

Priority:
5

Submitted By:
hub (khub)

Assigned To:
Bill Soudan (bills)

Category:
system

Status:
Closed

Summary:
crash with icq chat
each time I try an icq chat session the whole program closes itself immediately

Followups:

Message

```
Date: 2000-07-29 08:56
Sender: denis
Ok, since khub reported it works for him, I am closing this bug.

To robvnl I repeat:
"please try latest sources from CVS"...
```

```
Date: 2000-06-29 13:02
Sender: robvnl
Module Name: kicq
Latest release is 19991212.
is written here, or did I probably install a Beta version 19991212?
It would be great to can chat again!
```

```
Date: 2000-06-18 01:10
Sender: khub
I've try the latest version, it seems to work perfectly
That's marvellous...
```

```
Date: 2000-06-17 06:50
Sender: denis
Ok, lets try it one more time - WHAT VERSION OF KICQ do you use?
There was dramatic improvements in chat code since 1991212 beta,
so please try latest sources from CVS and report your comments
back.
```

```
Date: 2000-06-08 12:32
Sender: robvnl
Hi, I have the exact same problem. It doesn't make difference
wether I initiate or the other party initiates the chat. I use
RedHat 6.2 and compiled kicq with the export QTDIR=/usr/lib/qt-1.45
(the previous libs) because it needed it. Also it doesn't make
difference to run with the old or new QTDIR set. Sometimes the
chat request results in an user ABORTed at the other side and
I get USER HAS LEFT the chat. In case kicq realy crashes I have
a core dump.
```

```
Date: 2000-05-29 05:03
Sender: denis
What version do you try?
```

Fig. 1. Example bug report and followup messages (adapted from http://sourceforge.net/tracker/index.php?func=detail&aid=206585&group_id=332&atid=100332**)**

4. Analysis

In this section we discuss how we analyzed the data to develop the four measures introduced above. The analysis started with basis exploration of the data, which revealed problems with the quality of the data for some of the projects. For example, one team had consolidated bug reports from another bug tracking system into the SourceForge tracker. Unfortunately, these copied-over bugs all appeared in SourceForge to have been opened and closed within minutes, so this project was eliminated from further analysis.

4.1 Development team size

Since the FLOSS development process relies on contributions from active users as well as core developers, we sought a measure that reflected the size of this extended team, rather than just the core developers listed on the project page. As a proxy for the size of the extended development community, we counted the number of individuals who posted a bug or message to the bug tracker. As described above, the poster of bug reports and related messages are identified by a SourceForge ID (though postings can be anonymous), making it possible to count the number of distinct IDs appearing for each project. The counts were log transformed to correct skew.

4.2 Bug fixing time

To assess a project's performance in bug fixing, we examined how long it took the program to fix bugs by calculating the lifespan of each bug from report to close using the timestamps recorded by SourceForge. The most straightforward analysis would be to calculate each project's average bug-fixing time. However, this approach has several problems. First, the time taken to fix bugs is highly skewed, making an average unrepresentative. Second and more problematically, because not all bugs were closed at the time of our study, we do not always know the actual lifespan, but only a lower bound. This type of data is described as censored. Bugs are assumed to be reported at random times, making the censoring time (the point of our data collection) independent of the bug lifespan. As a result, the data exhibits "random type I right-censoring" [7]. Finally, analyzing only the average does not take into account available bug-level data. If there are differences between projects in the types of bugs reported (e.g., in their severity), then these differences could affect the average lifespan for a project.

Analysis of censored lifespan data involves a statistical approach known as survival or event history analysis. The basic idea is to calculate from the life spans the hazard function, which is the instantaneous probability of a bug being fixed at any point during its life. The hazard function can then be used as a dependent variable in a regression. Equivalently, the analysis can be done of the survival rate, which is the percentage of the bugs still open after a given lifespan.

In order to ensure that the hazard function is always positive, it is typical to regress on the log of the hazard function. Various functional forms can be assumed for the hazard function vs. time: for example, in an analysis of human death rates, the hazard function is generally assumed to rise over time as the person ages, while for an analysis of physical events, it might be assumed to be constant. Factors that increase or decrease the hazard rate can be added as additional variables in the regression.

For our initial purpose of developing a project-level measure of bug fixing effectiveness, we simply entered project as a factor in the hazard regression along with the bug priority, allowing us to compute a hazard ratio for each project (ratio because of the use of the log of the hazard). The hazard ratios are the regression weights in the hazard function for the dummy variable for each project, using the first project as the baseline. The analysis was performed using the R-Project statistical system (http://www.r-project.org/), specifically the psm function from the Survival and Design packages.

4.3 Activity level and number of downloads

The two remaining measures, activity level and number of downloads, were taken directly from SourceForge and so did not require further analysis, other than a log transformation of both to correct skew.

5. Results

In this section, we present the results of our analysis, starting with descriptive data about each measure and then considering relationships between measures.

5.1. Descriptive data

We obtained data on a total of 62,110 bug reports, an average of 509 per project. The median number of reports was 279, indicating a skewed distribution of bug report counts. We counted a total of 14,922 unique IDs, of whom 1,280 were involved in more than one project (one was involved in 8 of the projects in our sample). Table 1 presents descriptive data for the four measures analyzed. A plot of the survival rate over time for bugs is shown in Figure 2. The plot shows that bugs with higher priorities are generally fixed more quickly, as expected, but some bugs remain unfixed even after years.

We experimented with different functional forms for fitting the bug hazard rates. Somewhat surprisingly, the form that fitted best was an exponential, that is, one in which the hazard rate is not time varying. The R^2 for the fit was 0.51, which indicates a good fit. To assess the validity of the computed hazard ratios as a measure of bug-fixing performance, we compared the hazard ratios to median bug lifespan for each project. The high correlation between these values (r=0.88) suggests that the computed hazard ratio does capture the performance of the project team on fixing bugs. Of the two measures, the hazard ratio seems preferable for further analysis because it takes into account the open bugs and possible differences between projects in the mix of bug priorities experienced.

5.2 Relationships among success measures

Table 2 shows the correlations among the four measures proposed. The signs are all in the expected directions, since the community size and number of downloads should increase with success, while the activity rank

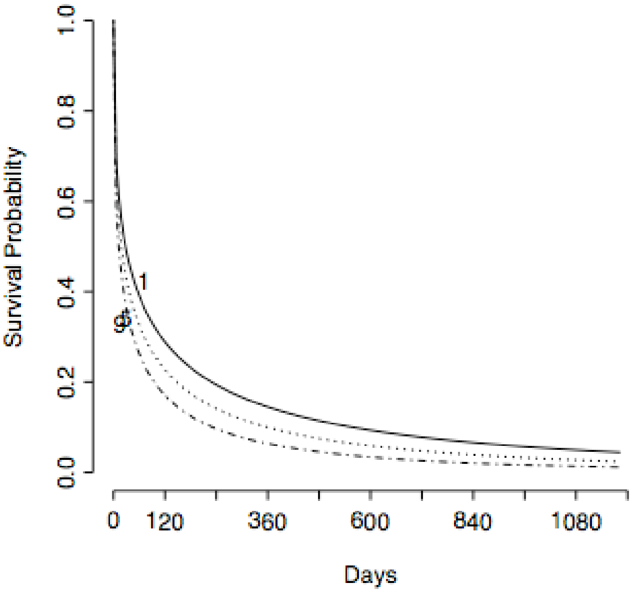


Figure 2. Plot of bug survival vs. time for high, default and low priority bugs.

	Mean	Median	Stdev
Community size	138.4	87.5	206
Activity rank	620	423	586
Hazard ratio	1.009	0.9438	1.06
Downloads	246,294	36,493	817,874

Table 1. Descriptive data for the four measures.

	CS	AR	HR	D
Community size	1.00	-0.73	-0.27	0.48
Activity rank	-0.73	1.00	0.14	-0.70
Hazard ratio	-0.27	0.14	1.00	0.06
Downloads	0.48	-0.70	0.06	1.00

Table 2. Correlations among measures.

and hazard ratio decrease (rank 1 is most active). Four of the six correlations are statistically significant (with 122 cases, the critical value for significance at p=0.05 is 0.17) indicating that there is a genuine relationship amongst them. The high correlations between activity rank, community size and number of downloads indicates that they are measuring a common factor, while the hazard ratio seems to be less connected.

6. Discussion

As mentioned in the introduction, the four measures applied in this paper have face validity as indicators. The preliminary analysis presented above allows us to extend our examination of the validity and utility of the measures.

The high correlation among these measures indicates a degree of convergent validity, since the different measures do correlate, particularly number of developers, activity and number of downloads. However, examining these correlations and correlations with other variables in more detail suggests room for improvement in the measures. In retrospect, our use of the SourceForge activity measure is problematic, since that activity is calculated using a formula that includes the number of bug reports and downloads. The high correlation of activity and downloads can thus be explained—they literally do measure the same construct.

The hazard ratio, on the other hand, is less strongly correlated, suggesting that it provides an independent view of a project's performance. This variable is most highly correlated with the size of the development community, which makes sense: more developers means that there is more help available to close bugs.

Another type of validity is predictive validity, meaning that the measures predict other variables of interest. The high correlations among variables are of little importance if those variables do not help us to understand the projects. We found that our proposed measure of project team size is highly correlated with the number of bug reports (r=0.68). The correlation between the age of a project and the number of bugs is low (r=0.09), suggesting that it is not simply the case that older projects have more time to collect reports and people. In other words, this correlation does seem to reflect a real phenomenon: more bug reports are indicative of a larger extended community. Interestingly, there is a negative correlation between the number of bugs and the time taken to fix them: projects with more developers report more bugs and then fix those bugs somewhat more quickly.

However, preliminary examination of the relationship between these measures and other data raise a second concern. In building a sample of projects to study, we seem to have found projects that all seem to be successful, by and large. As a result, the sample may not have sufficient variance on success, meaning that correlations between these success measures and other variables will be noise. To address this concern, we should collect data on a broader range of projects, including some that seem clearly to be unsuccessful.

7. Conclusions

This paper makes a contribution to the developing body of empirical research on FLOSS by identifying and operationalizing four success measures that might be applied to FLOSS. We have collected data on these measures for a set of SourceForge projects and shown the relations among these measures. We emphasize again that we do not view any single measure as the final word on success. As the measures draw on different aspects of the development process, they offer different perspectives on the process. Including multiple measures in a portfolio should provide a better assessment of the effectiveness of each project.

Having identified particular effective projects, our future work includes more detailed analysis of the projects. We plan to employ a theoretical sampling strategy to choose a few FLOSS development teams to study in depth. By limiting the number of projects, we will be able to use more labour-intensive data analysis approaches to shed more light on the practices of effect FLOSS teams.

8. References

[1] W. Scacchi, "Software Development Practices in Open Software Development Communities: A Comparative Case Study (Position Paper)," 2002.

[2] R. A. Ghosh, "Free/Libre and Open Source Software: Survey and Study. Report of the FLOSS Workshop on Advancing the Research Agenda on Free / Open Source Software," European Commission, http://www.infonomics.nl/FLOSS /report/workshopreport.htm, 2002.

[3] K. Crowston, H. Annabi, and J. Howison, "Defining Open Source Software project success," in Proceedings of the 24th International Conference on Information Systems (ICIS 2003). Seattle, WA, 2003.

[4] W. H. DeLone and E. R. McLean, "The DeLone and McLean model of information systems success: a ten-year update," J. Manage. Inform. Syst., vol. 19, pp. 9-30, 2003.

[5] W. H. DeLone and E. R. McLean, "Information Systems Success: The Quest for the Dependent Variable," Information Systems Research, vol. 3, pp. 60–95, 1992.

[6] S. Krishnamurthy, "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," University of Washington, Bothell, Bothell, WA May 2002.

[7] F. E. Harrell, Regression modeling strategies: With applications to linear models, logistic regression, and survival analysis. New York: Springer, 2001.