8

FLOSS Project Effectiveness Measures[1]

Kevin Crowston and James Howison

Introduction

In this chapter, we develop and illustrate measures of the effectiveness of FLOSS projects. FLOSS is a broad term used to embrace software that is developed and released under either a "free software" or an "open source" license. While the free software and the open source movements are distinct, both kinds of licenses allow users to obtain and distribute the software's original source without charge (software is "free as in beer") and to inspect, modify, and redistribute modifications to the source code. While the open source movement views these freedoms pragmatically (as a development methodology), the free software movement emphasizes the meaning of "free as in speech," which is captured by the French/Spanish *libre*, and one of their methods of supporting those freedoms is "copyleft," famously embodied in the General Public License, meaning that derivative works must be made available under the same license terms as the original. This chapter focuses on development practices in distributed work, which are largely shared across the movements. For example, many (though by no means all) FLOSS developers contribute to projects as volunteers without working for a common organization or being paid. We therefore use the acronym FLOSS to refer collectively to free/libre and open source software.

1 This research was partially supported by NSF Grants 03-41475, 04-14468 and 05-27457. Previous versions of this chapter have appeared as Crowston, K., Annabi, H. and Howison, J. (2003). Defining Open Source Software Project Success. In *Proceedings of the 24th International Conference on Information Systems (ICIS 2003)*, December, Seattle, WA; and Crowston, K., Howison, J. and Annabi, H. (2006). Information Systems Success in Free and Open Source Software Development: Theory and Measures. *Software Process—Improvement and Practice*, 11(2): 123–48.

It is important to develop measures of effectiveness 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 on the Internet, which is itself heavily dependent on FLOSS tools, but as Scacchi (2002a, p. 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." An 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 (Ghosh 2002). But to be able to learn from teams that are working well, we need to have a definition of working well.

In the following sections of the chapter, we will first discuss several measures of project effectiveness, and then the procedure we used to obtain data with which to operationalize these 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 then illustrate how these measures can be used to compare projects as part of a research study. We conclude with some suggestions for future research.

Measuring Project Effectiveness

The most commonly cited model for information systems success is DeLone and McLean (1992, 2002, 2003), shown in Figure 8.1. This model suggests six interrelated measures of success: system quality, information quality, use, user satisfaction, individual impact and organizational impact. Seddon (1997) proposed a related model that includes system quality, information quality, perceived usefulness, user satisfaction, and IS use. DeLone and McLean state that their model was built by considering "a process model [that] has just three components: the creation of a system, the use of the system, and the consequences of this system use" (2002, p. 87). We note that the measures included in their model focus on the use and consequences of the system and do not consider development. The choice of measures seems to be influenced by the relative ease of access to the use environment compared to the development environment

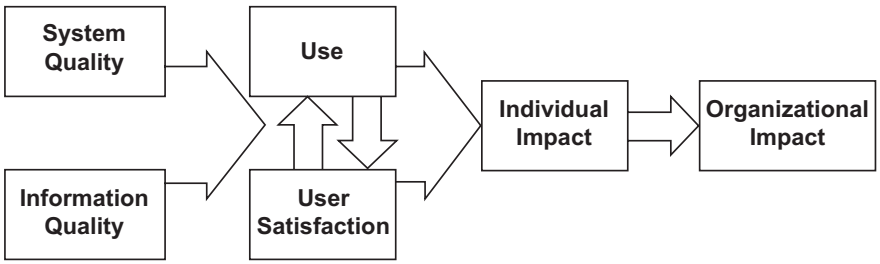

Figure 8.1 DeLone and McLean's model of information system success
Source: DeLone and McLean (1992, p. 87)

(especially true for packaged or commercial software). In the context of FLOSS though, researchers are frequently faced with the opposite situation, in that the development process is publicly visible and the use environment is difficult to study or even identify. We therefore start with a discussion of measures of success of the development process before considering the factors suggested by DeLone and McLean.

EFFECTIVENESS OF THE PROCESS OF SYSTEM DEVELOPMENT

For FLOSS projects, development is often considered an ongoing activity, as the project continues to release "often and early" (Raymond 1998). In other words, a FLOSS project is characterized by a continuing process of developers fixing bugs, adding features and releasing software. This characteristic of the FLOSS development process suggests a number of possible indicators of project effectiveness.

Number of Developers

First, since many FLOSS projects depend on volunteer developers, the ability of a project to attract and retain developers on an on-going basis is important for its success. Thus the *number of developers* involved in a project could be an indicator of success, both as an input for further develop and as an indirect measure of developer satisfaction with the project's processes and output. The number of developers can be measured in at least two ways. First, FLOSS development systems such as SourceForge (a free Web-based system that provides a range of tools to facilitate FLOSS development: http://sourceforge.net/) list *developers who are formally associated* with each project, meaning that they have been granted

permission to add code (to be a "committer") to the source code control system (e.g., CVS). In projects hosted elsewhere, this measure could be performed by examination of CVS logs to see which developers commit code. In both cases though, this count would give an underestimate of developers for projects in which code is generally contributed on a mailing list and integrated by a few developers, suggesting the need for caution in developing this measure.

Second, examination of the mailing lists and other fora associated with projects can reveal the *number of individuals who actively participate* in development activities without being formally a developer on the project. This measure can help gauge the level of involvement of the users as indicated by involvement of the users in submitting bug reports and participating in the project mailing lists, which is important because most FLOSS projects are dependent on help from users to identify problems, post suggestions, and even provide support for other users.

Level of activity

More important than the sheer number of developers is their contribution to a project. Thus the *level of activity* of developers in submitting code and bug reports may be useful as an indicator of project success. For example, SourceForge computes and reports a measure of project activity based on the activities of developers. Researchers could also examine development logs for evidence of software being written and released.

Cycle time

Another measure related to the group activity is *time between releases*. In FLOSS development, there is a strong community norm to "release early and release often", which implies that an active release cycle is a sign of a healthy development process and project. For example, FreshMeat (a web-based system that tracks releases of FLOSS: http://freshmeat.net/) provides a "vitality score" (Stewart and Ammeter 2002) that assesses how recently a project has made an announcement of progress on the FreshMeat site (http://freshmeat.net/faq/view/27/). (However, there is some suggestion that these measures are being gamed by developers anxious to see their project highly rated, again suggesting the need for caution in interpreting the numbers.)

More detailed examination of bug-fixing and feature-request fulfillment activities can yield useful process data indicative of the project's status. Bug

reports and feature requests are typically managed through a task-management system that records the developer and community discussion, permits labeling of priority items and sometimes includes informal "voting mechanisms" to allow the community to express its level of interest in a bug or new feature. The *time to close bugs* (or implement requested features) can be used as a measure of this aspect of project success.

PROJECT TEAM EFFECTIVENESS MEASURES

Finally, because the projects are ongoing, it seems important to consider the impact of a project on the abilities of the project team itself and its ability to continue or improve the development process. As Shenhar et al. (2001, p. 704) put it: "how does the current project help prepare the organization for future challenges?"

Employment opportunities

Some literature on the motivation of FLOSS developers suggests that developers participate to improve their employment opportunities (e.g., Lerner and Tirole 2000). Thus, one can consider *salary* (Hann et al. 2002) or *jobs acquired* through the involvement in a particular project as possible measures of success. For example, Hann et al. (2002) found that higher status within the Apache project was associated with significantly higher wages. Again, one might measure these indicators by surveying developers. While for a single developer, these measures are confounded with innate talent, training, luck, etc., aggregating across many developers and across time may provide a useful project-level measure of success.

Individual reputation

Similarly, literature also suggests that developers participating in FLOSS projects are rewarded with *reputation in the community*, and that this reputation is a sufficient reward for interaction. Kelty (2001) suggests that reputation might be measured through an analysis of credits located in source code (which he terms "greputation"). Alternative measures of FLOSS reputation might include the FLOSS communities' implementation of a "Web of Trust" at the community site Advogato (http://www.advogato.org/trust-metric.html) where developer status is conferred through peer review. Analyses of this kind of measure face the difficulty of tying the earning of reputation to the success of a particular project.

These measures might also be applied at the project level. Crowston, Howison and Annabi (2006) suggested *recognition* (e.g., mention on other sites) as a measure of project success. Similarly, another suggested measure was the *influence* of the product or project's process on other FLOSS groups and other commercial settings.

Knowledge creation

Projects can also lead to *creation of new knowledge* for individuals as well as on the group level (Arent and Nørbjerg 2000). Through their participation in a project, individual developers may acquire new procedural and programming skills that would benefit them on future projects. This effect could be measured by surveying the developers for their perceived learning.

In addition, following Grant's (1996) knowledge-based view of the firm, we view a firm (or in this case, a project) as a structure to integrate members' knowledge into products. In this view, the project's rules, procedures, norms, and existing products are a reflection of knowledge being created by the project activities. This *knowledge creation* can be measured by observing and qualitatively analyzing changes in the written rules and procedures over time and may be reflected and transferred through the development of systems for FLOSS project support, such as SourceForge and Savannah. Analysis of the development of interactions and support systems closely linked to a project might give some insight into this aspect of project success.

MEASURES OF THE OUTPUT OF SYSTEMS DEVELOPMENT

Two of the measures in the DeLone and McLean's model concern the product of the systems development process, namely systems quality and information quality. We first consider possible additional measures of this process step before turning to those measures.

Project completion

First, given the large number of abandoned projects (Ewusi-Mensah 1997), simply *completing a project* may be a sign of success. However, many FLOSS projects are continually in development, making it difficult to say when they are completed. Faced with this problem, Crowston and Scozzi (2002) instead measured success as the *progress of a project* from alpha to beta to stable status, as self-reported on SourceForge.

Second, another commonly used measure of success is whether the *project achieved its goals*. This assessment is typically made by a comparison of the project outcomes with the formal requirements specifications. However, FLOSS projects often do not have such specifications. Scacchi (2002b) examined the process of "requirements engineering" in open source projects and provided a comparison with the traditional processes (e.g., Jackson 1995; Davis 1990). He argues that rather than a formal process, FLOSS requirements are developed through what he terms "software informalisms", which do not result in agreed "requirements documentation" that could later be analyzed to see whether the project has met its goals. Scacchi's ethnography suggests that for FLOSS, goals will likely come from within through a discursive process centered on the developers. Therefore, a key measure for FLOSS may be simply *developer satisfaction* with the project, which could be measured by surveying developers. The developer community is much more clearly delineated than users, making such a survey feasible. Indeed, there have already been several FLOSS developer surveys (e.g., Ghosh 2002; Hertel et al. n.d.) though not on this topic specifically. Since in many projects there is a great disparity in the contribution of developers—a few developers contribute the bulk of the code (Mockus et al. 2000)—it may be desirable to weight developers' opinions in forming an overall assessment of a project.

System and information quality

Code quality has been studied extensively in software engineering. This literature provides many possible measures of the quality of software including understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness, and efficiency (Boehm et al. 1976; Gorton and Liu 2002). To this list might be added the *quality of the system documentation*. Code quality measures would seem to be particularly applicable for studies of FLOSS, since the code is publicly available. Indeed, a few studies have already examined this dimension. For example, Stamelos et al. (2002) suggested that FLOSS code is generally of good quality. Mishra, Prasad and Raghunathan (2002) offer an analytic model that suggests factors contributing to FLOSS code quality, such as number of developers, mix of talent level, etc. On the other hand, not many FLOSS systems include information per se, so the dimension of information quality seems to be less applicable.

MEASURES OF EFFECTIVENESS FROM THE USE OF A SYSTEM

User satisfaction

User satisfaction is an often-used measure of system success. For example, it is common to ask stakeholders if they felt a project was a success (e.g., Guinan et al. 1998). There is some data available regarding user satisfaction with FLOSS projects. For example, FreshMeat collects *user ratings* of projects. Unfortunately, these ratings are based on a non-random sample (i.e., users who take the time to volunteer a rating), making their representativeness suspect. Furthermore, we have observed that the scores seem to have low variance: in a recent sample of 59 projects, we found that scores ranged only from 7.47 to 9.07. It seems likely that users who do not like a piece of software simply do not bother to enter ratings. There do not seem to be any easily obtainable data on the related measures of perceived ease of use and usefulness (Davis 1989). *Opinions expressed on project mailing lists* are a potential source of qualitative data on these facets, though again there would be questions about the representativeness of the data.

In principle, it should be possible to *survey users* to collect their satisfaction with or perceptions of the software. However, to do so properly poses a serious methodological problem. Because most FLOSS projects are freely distributed through multiple channels, the population of users is unknown, making it impossible to create a true random sample of users. In this respect, FLOSS differs greatly from information systems developed in an organizational setting that have a clearly defined user population. The situation is also different than for the Web, another non-traditional systems environment, because with a Web site users are by definition the ones who visit the site, making the population effectively self-identifying. To achieve the same effect for FLOSS, the best solution might be to build the survey into the software, though doing so might annoy some users.

Use

Although there is some debate about its appropriateness (DeLone and McLean 2003; Seddon 1997), many studies employ system use as an indication of information systems success. For software for which use is voluntary, as is the case for most FLOSS, use seems like a potentially relevant indicator of the project's success. Unfortunately, actual usage data are available for only a few FLOSS projects. For example, Netcraft conducts

a survey of Web server deployment (http://news.netcraft.com/archives/webserver_survey.html), which estimates the market share of different Web servers. Other projects that require some kind of network connection could potentially be measured in the same way (e.g., instant messaging or peer-to-peer file sharing clients), but this approach does not seem to be widely applicable. Many programs do check regularly for updated versions, which would provide a good measure of actual use, but these numbers do not seem to be publicly available.

Popularity

Rather than measuring actual use, it may be sufficient to count the actual or potential *number of users* of the software, which we label "popularity" (Stewart and Ammeter 2002). A simple measure of popularity is the *number of downloads* made of a project. These numbers are readily available from various sites. Of course not all downloads result in use, so variance in the conversion ratio will make downloads an unreliable indicator of use. Furthermore, because FLOSS can be distributed through multiple outlets, online as well as offline (e.g., on CDs), the count from any single source is likely to be quite unreliable as a measure of total users. A particularly important channel is "distributions" such as RedHat, SuSE or Debian. Distributions provide purchasers with pre-selected bundles of software packaged for easy installation and are often sold on a CD-ROM to obviate the need to download everything. Indeed, the most popular software might be downloaded only rarely because it is already installed on most users' machines and stable enough to not require the download of regular updates. Therefore, an important measure of popularity to consider is the package's *inclusion in distributions*. Crowston et al. (2006) found that developers consider porting of a product to different systems (especially to Windows), and requests for such ports as a measure of the success of the product. This theme might be considered a special case of popularity.

Other sources of data reflecting on users are available. Freshmeat provides a *popularity* measure for packages it tracks, though a better name might be "interest", as it is one step further removed from actual use. The measure is calculated as the geometric mean of subscriptions and two counts of page viewings of project information. Similarly, SourceForge provides information on the number of *page views of the information pages* for projects it supports.

Finally, it may be informative to measure use from perspectives other than that of an end user. In particular, the openness of FLOSS means that other

projects can build on top of it. Therefore, one measure of a project's success may be that many other projects use it. *Package dependency* information between projects can be obtained from the package descriptions available through the various distributions' package management systems. Analysis of source code could reveal the *reuse of code* from project to project (though identifying the origin could be difficult).

Individual or organizational impacts

The final measures in DeLone and McLean's (1992) model are individual and organizational impacts for the users. Though there is considerable interest in the economic implications of FLOSS, these measures are hard to define for regular information systems projects and doubly hard for FLOSS projects, because of the problems defining the intended user base and expected outcomes. Therefore, these measures are likely to be unusable for most studies of individual FLOSS projects.

SourceForge Data on Effectiveness

In this chapter, we illustrate the use of a portfolio of measures in two ways. Specifically, we analyze four possible success measures, namely number of developers, number of mailing list participants, and number of downloads and page views for SourceForge projects in general and for a smaller sample of projects in more detail. These measures were chosen from the list above because they included both inputs (number of developers and users) and outputs (number of downloads and project Web page views).

Each of the proposed measures has good face validity, in the sense that a project that attracts developers and that many users download should be described as a success, especially if it continues to do so over time. However, we are 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 this section, we illustrate these measures using data about a large sample of FLOSS projects at a single point in time. To create a sample of FLOSS projects, we selected from projects hosted by SourceForge. As of May 2007, SourceForge claimed nearly 150,000 FLOSS projects on a wide diversity of topics.

Table 8.1 Descriptive statistics for selected effectiveness measures for the full sample of projects

Variable	Mean	Median	Standard deviation
downloads	10,760	52	423,103
page views	75,908	628	3,564,219
developers	2.35	1	4.45
unique message authentication	1.59	1	5.36

N = 65,070

Data for the four measures adopted are tracked by SourceForge and available for research from the SourceForge Research Data Archive (http://www.nd.edu/~oss/) and from the FLOSSmole project (http://ossmole.sourceforge.net; Howison et al. 2006). We obtained downloads, page views, and number of unique users in messages from the April 2007 dump in the SourceForge Research Data Archive (table stats_project_all). The developer count in this table counts active developers (though the definition of active is not given), so we obtained the total listed developer count from the April 2007 spider run in FLOSSmole. Projects that could not be spidered by FLOSSmole were eliminated from the sample. NULL values in the database dumps were replaced with 0s. The full dataset included 65,070 projects.

Clearly not all of these projects were suitable for our study: many are inactive and previous studies have suggested that many are in fact individual projects (Krishnamurthy 2002), as is borne out by the median of one developer. The low median number of downloads suggests that many of these projects are not actively distributing code. Since we are primarily interested in development practices in distributed groups, we restricted our analysis to projects that listed more than seven developers and had any downloads as of the date of the study. 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 downloads is indicative of a minimal level of development effort and having released files. Only 2,168 projects in our dataset satisfied these two criteria. Table 8.2 shows the descriptive statistics for the four measures. The large difference between the mean and median for these variables indicates that they are heavily skewed. The skew can be largely corrected for downloads and page views by applying a log transformation, as shown by the histograms of these measures in Figure 8.2. However, the other two variables remain heavily skewed even with such a transformation. (The skew in developer counts is even more pronounced for the entire dataset, so it is not a result of the sampling.)

Table 8.2 Descriptive statistics for selected effectiveness measures

Variable	Mean	Median	Standard deviation
downloads	138,818.4	5,316.5	1,225,832
log downloads	3.68	3.73	1.17
page views	868,641.2	46,131.5	8,643,556
log page views + 1	4.62	4.66	1.18
developers	15.87	12	16.02
log developers	1.12	1.08	0.22
unique message authentication	6.67	1	23.68
log unique message authentication + 1	0.54	0.30	0.40

N = 2,168

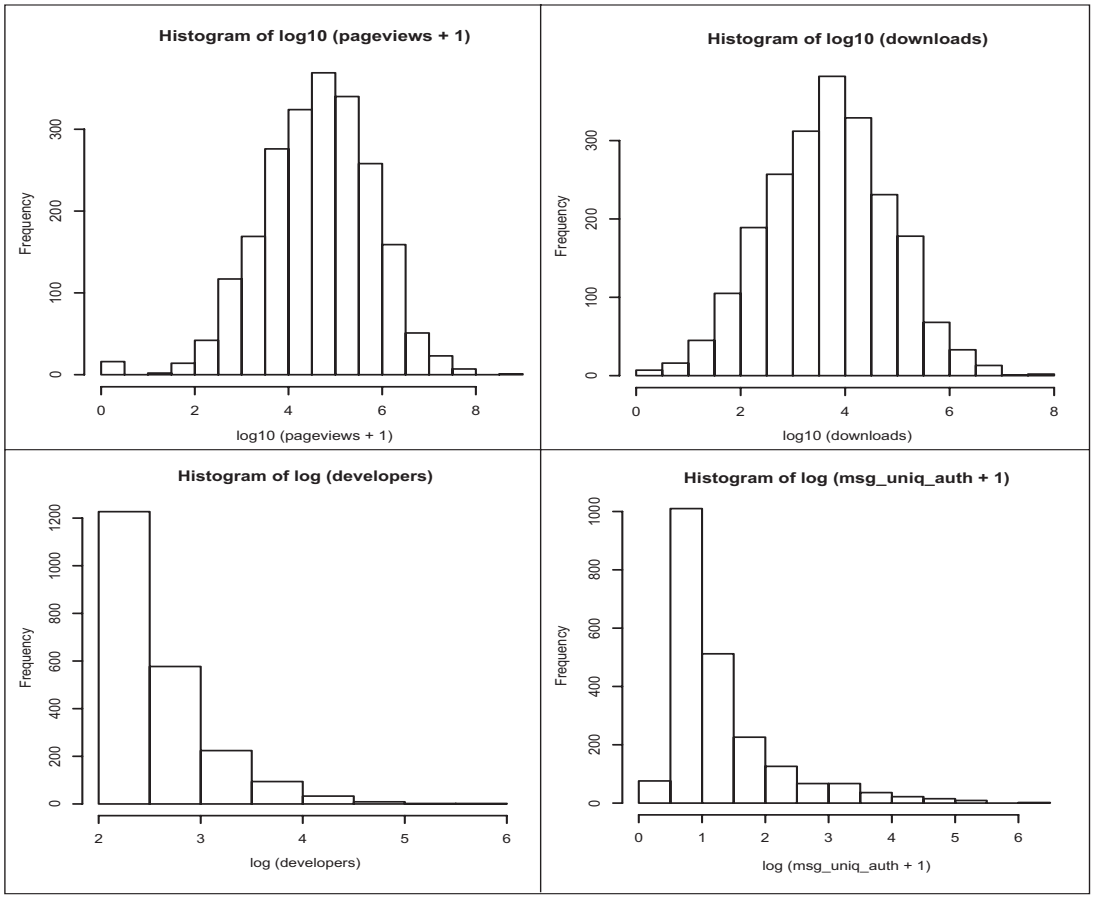

Figure 8.2 Distributions of selected variables (log transformed)

We next examine the relation among these variables. Because the data are not normally distributed, we examined the relation using non-parametric correlation, namely Spearman's R (results using Kendall's tau are similar). The correlations are shown in Table 8.3. These correlations suggest that downloads and page views are closely related, so both indicate a project's overall popularity. Interestingly, the level of participation in the messages seems to be more closely related to popularity than to the number of developers, suggesting this measure as a way to gauge the broader community around a project.

For comparison, Table 8.4 shows the correlations calculated for the entire dataset. The correlation for downloads is about the same, but note that including all projects boosts the correlation between the number of developers and number of posters of messages, likely because of the large number of projects scoring very low on both measures. This result underscores the importance of developing a sample that properly captures the phenomenon of interest (in our case, distributed software development rather than simply creation of a SourceForge project).

Table 8.3 Spearman's R correlations among variables for active projects

	Downloads	Page views	Developers
page views	0.75		
developers	0.21	0.25	
unique message authentication	0.30	0.28	0.06

N = 2,168

Table 8.4 Spearman's R correlations among variables for all projects

	Downloads	Page views	Developers
page views	0.74		
developers	0.18	0.33	
unique message authentication	0.29	0.35	0.22

N = 65,070

Case Study

In this section we present an example of how these measures might be used to compare the effectiveness of projects as a dependent variable in a study. We compare six FLOSS projects chosen to allow for meaningful comparisons. First,

we controlled for topic. Projects within a single topic category are potential competitors so making comparisons of outcomes such as downloads between these projects valid. Second, to minimize unwanted variance, we chose projects that are roughly similar in age and status (production/stable). Projects at this stage have relatively developed membership and sufficient team history, yet the software code still has room for improvement, which enables us to observe rich team interaction processes. On the other hand, we wanted to have projects at different levels of complexity to provide for variability. Accordingly we picked three projects that develop enterprise resource planning (ERP) systems (Compiere, WebERP, and Apache OFBiz) and three teams that develop Instant Messenger (IM) clients (Gaim, aMSN, and Fire). ERP projects are more complex than IM projects since they have to address many external constraints such as accounting rules and legal reporting requirements.

The array of measures presented in Figure 8.3 and Figure 8.4 use data collected by the FLOSSmole project (Howison et al. 2006) from the project establishment in SourceForge until around March 2006. Note that we have taken advantage of the rich data available to show the evolution of these measures over time, rather than comparing all time measures. This comparison suggests that the most effective IM project is Gaim, followed by aMSN, then Fire, and the most effective ERP project is Compiere followed by OFBiz then WebERP. Further study can then address the question of what practices adopted by these different projects seem to be related to these differences in effectiveness. For example, what (if anything) are Gaim developers doing that makes the project more attractive to other developers or the resulting program more attractive to users?

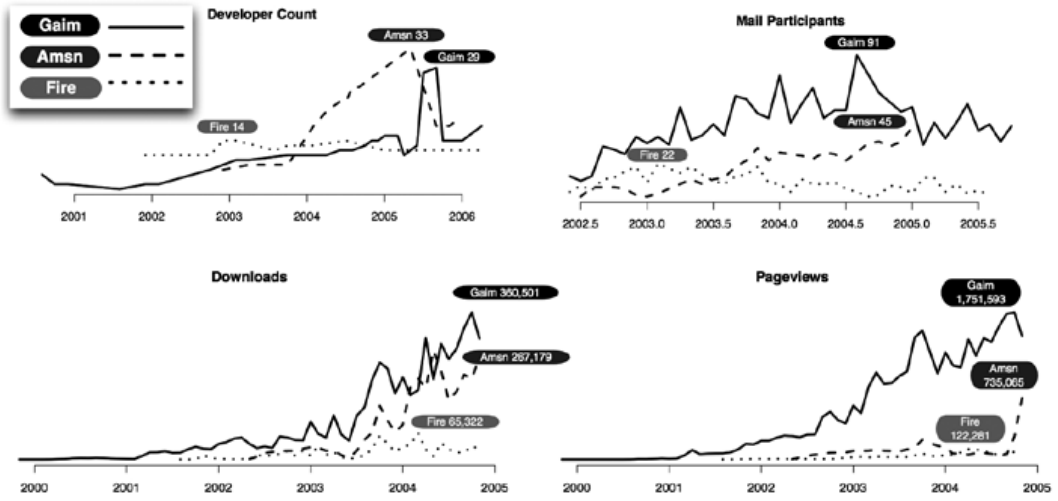

Figure 8.3 Comparison of effectiveness measures for IM projects

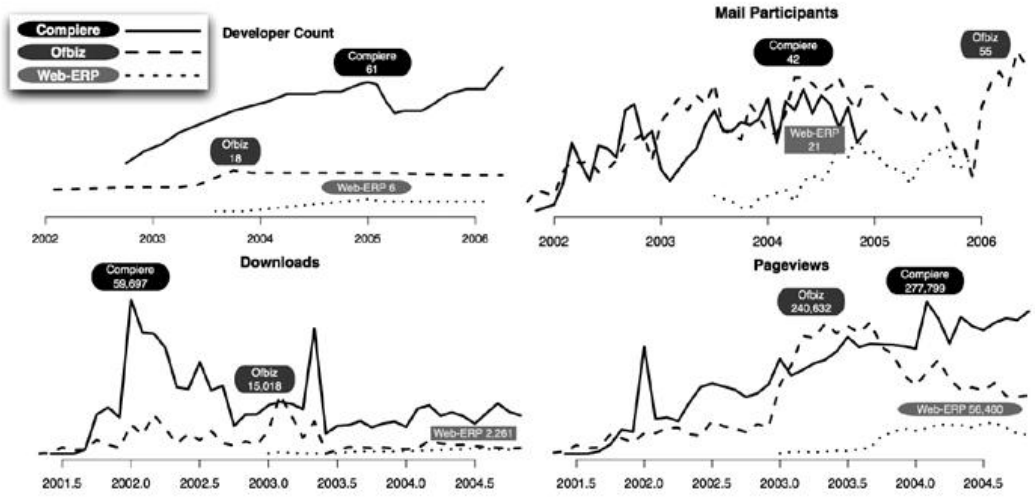

Figure 8.4 Comparison of effectiveness measures for ERP projects

Conclusions

This chapter makes a contribution to the developing body of empirical research on FLOSS by identifying a collection of success measures that might be applied to FLOSS. We have collected data on four possible measures for a set of SourceForge projects and shown the relations among these measures. The four measures applied in this chapter all have face validity as indicators. We emphasize again that we do not view any single measure as the final word on success. The moderate correlation among these measures indicates a degree of convergent validity, but 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.

Most of the constructs considered in this chapter can be represented as cumulative variables, as in the large sample study above, or as time-series, as in our case study above. Even though the required analyses are more complex, there are clear reasons to prefer the time-series representation over the cumulative representation, especially as analysis moves to understanding the sources of project effectiveness. The underlying logic here is that the practices and structures reflected in the effectiveness measures may need to vary over time if projects are to successfully meet the challenges of different phases of the project's development (e.g., Rajlich and B.K.H. 2000). For example, initially it

may be important to have a quite small group of developers, who can lay down a coordinated and consistently designed artifact, perhaps with appropriate modularity. Later, that small-team design may be better able to support a larger and rapidly growing group of developers (Senyard and Michlmayr 2004; Parnas et al. 1981). If such processes are at play, different measures may be appropriate for projects at different stages of growth. Furthermore there may be multiple routes to eventual success, and clustering time-series is a useful approach to understanding such paths (Stewart et al. 2006).

Time-series also offer the possibility of identifying periods of particular interest in the life of a project. For example, one can identify transition points through techniques such as interrupted time-series experiments. These techniques make it possible to identify projects that appeared to be performing well (on a combination of measures) but then stall, or projects that appeared to have stalled but manage to re-start and return to effective operation. The path of aMSN's developer count in Figure 8.4 is a suggestive example, with two seeming transition points. Such transition points could highlight periods for in-depth analysis which may cast detailed light on project risks and effective coping strategies.

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 labor-intensive data analysis approaches to shed more light on the practices of effective FLOSS teams.

References

Arent, J. and Nørbjerg, J. (2000) Software Process Improvement as Organizational Knowledge Creation: A Multiple Case Analysis. Presented at Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS-33, January 4–7), Wailea, Maui, HI.

Boehm, B.W., Brown, J.R. and Lipow, M. (1976) Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering*, October 13–15. San Francisco, CA: 592–605.

Crowston, K., Howison, J. and Annabi, H. (2006) Information Systems Success in Free and Open Source Software Development: Theory and Measures, *Software Process—Improvement and Practice*, 11: 123–48.

Crowston, K. and Scozzi, B. (2002) Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings Software*, 149: 3–17.

Davis, F.D. (1989) Perceived Usefulness, Perceived Ease of Use and User Acceptance of Information Technology, *MIS Quarterly*, 13: 319–40.

Davis, A.M. (1990) *Software Requirements Analysis and Specification.* Englewood Cliffs, NJ: Prentice-Hall.

DeLone, W.H. and McLean, E.R. (1992) Information Systems Success: The Quest for the Dependent Variable, *Information Systems Research*, 3: 60–95.

DeLone, W.H. and McLean, E.R. (2002) Information Systems Success Revisited. Presented at Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35, January 7–10), Waikoloa, Hawaii.

DeLone, W.H. and McLean, E.R. (2003) The DeLone and McLean Model of Information Systems Success: a Ten-year Update, *J. Manage. Inform. Syst.*, 19: 9–30.

Ewusi-Mensah, K. (1997) Critical Issues in Abandoned Information Systems Development Projects, *Communication of the ACM*, 40: 74–80.

Ghosh, R.A. (2002) Free/Libre and Open Source Software: Survey and Study. Report of the FLOSS Workshop on Advancing the Research Agenda on Free/Open Source Software, online document. http://www.flossproject.org/report/FLOSS_Final5all.pdf. Last accessed on October 19 2010.

Gorton, I. and Liu, A. (2002) Software Component Quality Assessment in Practice: Successes and Practical Impediments. In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*. May. Orlando, FL: Association for Computing Machinery (ACM) Press, 555–8.

Grant, R.M. (1996) Toward a Knowledge-based Theory of the Firm, *Strategic Management Journal*, 17: 109–22.

Guinan, P.J., Cooprider, J.G. and Faraj, S. (1998) Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach, *Inf. Syst. Res.*, 9: 101–25.

Hann, I.-H., Roberts, J., Slaughter, S. and Fielding, R. (2002) Economic Incentives for Participating in Open Source Software Projects, In *Proceedings of the Twenty-Third International Conference on Information Systems*, Seattle WA: 365–72.

Hertel, G., Niedner, S. and Herrmann, S. (n.d.) Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel. Kiel, Germany: University of Kiel.

Howison, J. Conklin, M. and Crowston, K. (2006) FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses, *International Journal of Information Technology and Web Engineering*, 1: 17–26.

Jackson, M. (1995) Software Requirements and Specifications: Practice, Principles, and Prejudices. Boston, MA: Addison-Wesley.

Kelty, C. (2001) Free Software/Free Science, First Monday, 6(12). http://firstmonday.org/issues/issue6_12/kelty/index.html. Last accessed October 19 2010.

Krishnamurthy, S. (2002). Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday*, 7(6). http://131.193.153.231/www/issues/issue7_6/krishnamurthy/index.html. Last accessed on October 19 2010.

Lerner, J. and Tirole, J. (2002). The Scope of Open Source Licensing, working papers, http://idei.fr/doc/wp/2003/scope_open_source.pdf . Last accessed on October 19 2010.

Mishra, B., Prasad, A. and Raghunathan, S. (2002) Quality and Profits Under Open Source Versus Closed Source. In *Proceedings of the Twenty-Third International Conference on Information Systems*: December, Barcelona, Spain.

Mockus, A., Fielding, R.T. and Herbsleb, J.D. (2000) A Case Study of Open Source Software Development: The Apache Server. Presented at Proceedings of the International Conference on Software Engineering (ICSE'2000). June, Limerick, Ireland.

Parnas, D.L., Clements, P.C. and Weiss, D.M. (1981) The Modular Structure of Complex Systems, *IEEE Transactions on Software Engineering*, 11: 259–66.

Rajlich, V.T. and B.K.H. (2000) A Staged Model for the Software Life Cycle, *IEEE Computer*, 33: 66–71.

Raymond, E.S. (1998) The Cathedral and the Bazaar, *First Monday*, 3,3. http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/. Last accessed October 19 2010.

Scacchi, W. (2002a) Software Development Practices in Open Software Development Communities: A Comparative Case Study. Position paper for the 1st workshop on Open Source Software Engineering, Toronto, Ontario. http://opensource.ucc.ie/icse2001/scacchi.pdf. Last accessed October 19 2010.

Scacchi, W. (2002b) Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings Software*, 149: 24–39.

Seddon, P.B. (1997) A Respecification and Extension of the DeLone and McLean Model of IS Success, *Information Systems Research*, 8: 240–53.

Senyard A. and Michlmayr, M. (2004) How to Have a Successful Free Software Project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, 30 November–3 December, Busan Korea: 84–91.

Shenhar, A.J., Dvir, D., Levy, O. and Maltz, A.C. (2001) Project Success: A Multidimensional Strategic Concept, *Long Range Planning*, 34: 699–725.

Stamelos, I., Angelis, L. Oikonomou, A. and Bleris, G.L. (2002) Code Quality Analysis in Open Source Software Development, *Information Systems Journal*, 12: 43–60.

Stewart, K.J. and Ammeter, T. (2002) An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects. In L. Applegate, R.Galliers, and J.I. DeGross (eds), *Proceedings of the Twenty-Third International Conference on Information Systems*. Barcelona, Spain: 853–7.

Stewart, K.J., Darcy, D.P. and Daniel, S. (2006) Opportunities and Challenges Applying Functional Data Analysis to the Study of Open Source Software, *Statistical Science*, 21(2): 167–78.