SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2006; **11**: 123–148
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.259

Information Systems Success in Free and Open Source Software Development: Theory and Measures[‡]

Research Section

Kevin Crowston[1*,†], James Howison[1] and Hala Annabi[2]
[1] *Syracuse University, School of Information Studies, 348 Hinds Hall, Syracuse, NY 13244-4100, USA*
[2] *University of Washington, The Information School, Box 352840, Suite 370 Mary Gates Hall, Seattle, WA 98195-2840, USA*

Information systems success is one of the most widely used dependent variables in information systems (IS) research, but research on free/libre and open source software (FLOSS) often fails to appropriately conceptualize this important concept. In this article, we reconsider what success means within a FLOSS context. We first review existing models of IS success and success variables used in FLOSS research and assess them for their usefulness, practicality and fit to the FLOSS context. Then, drawing on a theoretical model of group effectiveness in the FLOSS development process, as well as an on-line discussion with developers, we present additional concepts that are central to an appropriate understanding of success for FLOSS.

In order to examine the practicality and validity of this conceptual scheme, the second half of our article presents an empirical study that demonstrates operationalizations of the chosen measures and assesses their internal validity. We use data from SourceForge to measure the project's effectiveness in team building, the speed of the project at responding to bug reports and the project's popularity. We conclude by discussing the implications of this study for our proposed extension of IS success in the context of FLOSS development and highlight future directions for research. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: free/libre open source software; information systems success; concept development; survival analysis

[*] Correspondence to: Kevin Crowston, Syracuse University School of Information Studies, 348 Hinds Hall, Syracuse, NY 13244-4100 USA
[†]E-mail: crowston@syr.edu
[‡]Earlier versions of this article appeared as: Crowston, K., Annabi, H., & Howison, J. (2003): Defining open source software project success, iI in Proceedings of the 24th International Conference on Information Systems (ICIS 2003), Seattle, WA; Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004): Towards a portfolio of FLOSS project success measures, Article presented at the Workshop on Open Source Software Engineering, 26th International Conference on Software Engineering, Edinburgh.

1. INTRODUCTION

The long-term goal of our research is to identify processes that enable distributed software team performance, specifically, the performance of free/libre open source software (FLOSS) development teams. In this article, we take a needed step in this direction by developing measures for the success of FLOSS projects. This step is needed because we will not be able to improve software processes if we cannot identify what constitutes an improvement. Information systems (IS) success is one of the most widely used dependent variables in IS research. Not surprisingly, much attention has been given to how best to measure it (e.g. DeLone and McLean 1992, 2002, 2003, Seddon *et al.* 1999, Rai *et al.* 2002, Seddon 1997). However, the unique nature of FLOSS development makes some measures more appropriate than others and requires the addition of hitherto unconsidered measures.

FLOSS is a broad term used to embrace software that is developed and released under either a 'free software' or an 'open source' license.[1] Both open source and free software are free in two senses: 'free as in speech', meaning that the code may be redistributed and reused in other FLOSS projects, and 'free as in beer', meaning that the software is available for download without charge. As well, many (though by no means all) FLOSS developers contribute to projects as volunteers without working for a common organization or being paid. As we will discuss, these two characteristics have implications for the applicability of certain measures of success.

It is important to develop measures of success for FLOSS projects for at least two reasons. First, having such measures will be useful for FLOSS project leaders 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, including major corporations, depend on FLOSS systems such as Linux (and, of course, the Internet, which is heavily dependent on FLOSS tools), but as Scacchi (2002) 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'.

1.1. Outline of Article

This search for appropriate FLOSS success measures presented in this article proceeds according to the following outline. In the first half of the article, we develop a richer conceptualization of success measures for FLOSS drawing on a number of sources. We first review the literature on IS success to see what measures might be adopted and to identify problems in applying others to the context of FLOSS development. We then step back and discuss the process model underlying the existing IS models and extend these models to fit the FLOSS context. We do so with reference to a model of group effectiveness derived from Hackman (1987). We next assess the face validity of our conceptual scheme using the opinions of FLOSS developers elicited through SlashDot, a popular Web-based discussion board (http://slashdot.org/). The comparison suggests additional measures that might be incorporated to develop a fuller understanding of FLOSS project success that we integrate into our conceptual scheme of success in FLOSS development. Finally, we examine recent research articles on FLOSS to see what measures of success have been used in practice, and comment on their appropriateness and utility. The result of this conceptual development work is a set of possible measures

[1] The free software movement and the open source movement are distinct but share important characteristics. The licenses they use 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. See http://www.gnu.org/philosophy/and http://opensource.org. While the differences and similarities of the movements are interesting, this article focuses on development practices in distributed work and those are largely shared across the movements. We therefore choose the acronym FLOSS, standing for free/libre and open source software.

of FLOSS development effectiveness and related operationalizations.

In order to examine the practicality and validity of this conceptual scheme, in the second half of our article we present an empirical study that demonstrates its operationalization and assesses the internal validity of the measures. For this purpose, we use data from SourceForge, the largest hub for FLOSS development projects. Finally we conclude by discussing the implications of this study for our proposed extension of IS success in the context of FLOSS development and highlight future directions for research.

2. THEORY DEVELOPMENT: MEASURING THE SUCCESS OF FLOSS DEVELOPMENT

In this section, we describe the process through which we developed a conceptual model of success measures for FLOSS development. We discuss in turn our review of models of success in the IS literature, extensions to existing conceptual models, feedback from FLOSS developers and review of measure of success applied in the empirical FLOSS literature.

2.1. Literature Review: Conceptual Models of Information System Success

FLOSS is a form of system development, so we begin our hunt for success measures in the IS literature. Note though that we are not attempting an exhaustive review of this extensive literature, but rather are using the conceptual models presented in the literature to identify success measures relevant to FLOSS. The most commonly cited model for IS success is that of DeLone and McLean (1992, 2002, 2003), shown in Figure 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. Taken together, these models suggest a number of possible measures that could be applied to FLOSS.

2.1.1. 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). ISO standard 9126 defines software quality as including functionality, reliability, usability, efficiency, maintainability and portability, each with subdimensions. A commonly used measure of quality is the *number of defects per thousand lines of code* (Diaz and Sligo 1997, Goranson 1997) or the probability of a fault in a module (Basili *et al.* 1994). To this list should be added the *quality of the system documentation*. Code quality measures would seem to be particularly practical 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 *et al.* (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 (i.e. data) *per se*, so the dimension of information quality seems to be less applicable.

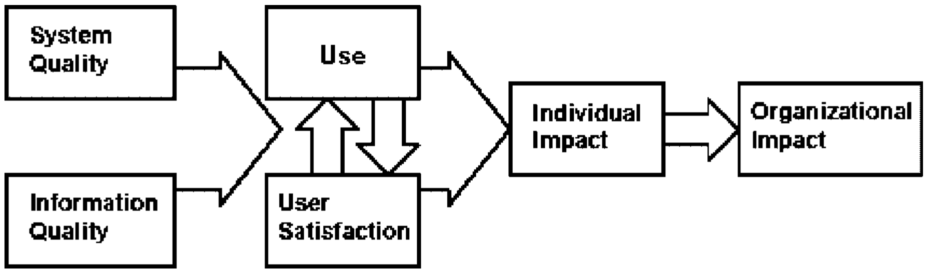


Figure 1. DeLone and McLean's Model of IS Success (DeLone and McLean (1992), Figure 2, p. 87) ''Reprinted by permission. Copyright 1992 INFORMS. DeLone and McLean ER. 1992. Information Systems Success. The quest for the dependent variable *Information Systems Research* **3**(1): 60–95, The institute for Operations Research and the Management Sciences, 7240 Parkway Drive, Suite 310, Hanover, Maryland 21076, USA''

2.1.2. *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, a Web-based system that tracks releases of FLOSS (http://freshmeat.net/), collects *user ratings* of projects. Unfortunately, these ratings are based on a nonrandom 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 out of 10. 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 IS developed in an organizational setting or for the commercial market, which have a clearly defined user population. The situation is also different from that for the Web, another nontraditional 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. For example, recent versions of the Mozilla Web browser include a program that offers to report crashes and collect other feedback.

2.1.3. *Use*

Although there is some debate about its appropriateness (DeLone and McLean 2003, Seddon 1997), many studies employ *system use* as an indication of IS 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. Some interesting data are available. For rare projects, these numbers can be directly measured. For example, Netcraft conducts a survey of Web server deployment,[2] 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. Avery Pennarun's Debian Popularity Contest[3] collects statistics on the usage of software on Linux machines running the Debian distribution. Users install a program that collects and reports usage information daily and the resulting statistics show that packages have been installed, and which of these have been recently used. Unfortunately, these data are collected from a nonrandom sample of machines, running a particular Linux distribution, so the results are likely not representative of use in the broader population.

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, and a popular one in the FLOSS literature reviewed below, is the *number of downloads* made of a project. Download 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, on-line 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*.

[2] http://news.netcraft.com/archives/webserver_survey.html

[3] http://people.debian.org/~apenwarr/popcon/

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.[4] 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 from 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 true origin of the code could be difficult).

2.1.4. 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 I/S 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.

[4] http://freshmeat.net/faq/view/30/

2.1.5. Summary

To summarize, existing models of IS success suggest a range of potential success measures for FLOSS projects as shown in Table 1. However, a number of the measures are inapplicable, while others are difficult to apply in the FLOSS environment. We note that many of these measures are based on a vision of system development in an organization and do not take into account the unique characteristics of the FLOSS development environment. A deeper understanding of the differences between the process model underlying the IS success literature and the process of FLOSS development is needed.

2.2. Reconsidering Process: the Process of Floss Development

The previous section considered how success has been measured in the IS literature and its applicability to FLOSS development. In this section, we extend these models by re-examining the vision of systems development underlying DeLone and McLean's success model to identify additional measures that might be used for FLOSS project success. DeLone and McLean explicitly state that their model

Table 1. Success measures suggested by the IS literature

Measure of success	Indicators	Audience
System and information quality	Code quality (e.g. understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness, efficiency) Documentation quality	Users, developers
User satisfaction	User ratings Opinions on mailing lists User surveys	Users, developers
Use	Use (e.g. Debian popularity contest) Number of users Downloads Inclusion in distributions Popularity or views of information page Package dependencies Reuse of code	Developers
Individual and organizational impacts	Economic and other implications	Users, developers

of project success 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' (DeLone and McLean 2002), which we have shown graphically in Figure 2. We note that the measures included in the model focus on the use and consequences of the system (the right side of the figure), and do not open up either box in the process. While this focus may be appropriate given the traditional concern of IS research with the organizational implication of IS, it seems to unduly restrict the range of measures considered.

The choice of measures also seems to be influenced by the relative ease of access to the use environment compared to the development environment for packaged or commercial software. In the context of FLOSS though, researchers are frequently faced with the opposite situation, in that many aspects of the development process are publicly visible while the use environment is difficult to study or even identify. For both reasons, we believe that it will be useful to complement previously identified IS success measures with the ones that take advantage of the availability of data on the development process. As a structure for analyzing the systems development process, we draw on Hackman's (1987) model of effectiveness of work teams. An attractive feature of this model is that effectiveness is conceptualized along multiple dimensions. In addition to *task output*, Hackman includes the *team's continued capability to work together* and *satisfaction of individual team members' personal needs* as relevant outputs. The following discussion examines such measures of success.

2.2.1. *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 in the FLOSS context.

Project Completion. First, given the large number of abandoned software projects (Ewusi-Mensah 1997), simply *completing a project* may be a sign of success. However, it is common for FLOSS projects to be 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 by the team. For example, for many teams the 1.0 release is a significant milestone.

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 (2002) examined the process of 'requirements engineering' in open source projects and provided a comparison with the traditional processes (e.g. Davis 1990, Jackson 1995). 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 further suggests that for FLOSS, goals will likely come from within the project through a discursive process centered on the developers. Therefore, a key measure for FLOSS may be simply *developer satisfaction* with the project, which corresponds to Hackman's (1987) individual satisfaction dimension for group performance. Developer satisfaction could be measured by surveying developers: the developer community is much more clearly

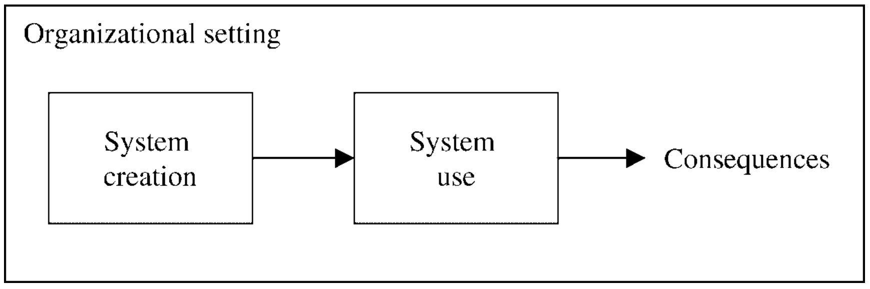


Figure 2. Process model underlying the DeLone and McLean (1992) model of success

delineated than users, making such a survey feasible. Indeed, there have already been several FLOSS developer surveys (e.g. Ghosh 2002, Hertel *et al.* 2004), 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.

2.2.2. Measures of the Process of Systems Development

In DeLone and McLean's (1992) process model, systems development is implicitly treated as a one-off event. However, for FLOSS projects (and indeed many other types of projects) development is instead an ongoing activity, as the project continues to release 'often and early' (Raymond 1998). In other words, a FLOSS project is often characterized by a continuing process of developers fixing bugs, adding features and releasing new versions of the software. This characteristic of the FLOSS development process suggests a number of possible indicators of success.

Number of Developers. First, since many FLOSS projects are dependent on volunteer developers, the ability of a project to attract and retain developers on an ongoing basis is important for its success. Thus the *number of developers* involved in a project could be an indicator of success. The number of developers can be measured in at least two ways. First, FLOSS development systems such as SourceForge list *developers who are formally associated* with each project, a measure that could also be discovered through examination of Concurrent Versions System (CVS) logs for projects where developers contribute code directly (i.e. not via a mediated patch submission process). 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 associated with the project.

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* on the basis of 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 provides a *vitality score* (Stewart and Ammeter 2002) that assesses how recently a project has made an announcement of progress on the Freshmeat site.[5] In addition, detailed examination of bug-fixing and feature-request fulfillment activities might yield useful process data indicative of the project's status. These processes involve interaction with the user community and can involve applying patches of contributed code supplied by noncore developers. Bug reports and feature requests are often managed through a task-management system that records the developer and community discussions and 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) and the *proportion of bugs fixed* therefore might be helpful measures of the strength of a project's processes and thus indirectly of its success.

2.2.3. Effects on Project Teams

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. This dimension corresponds to the Hackman's (1987) dimension of the team's continued capability to work together as a measure of team performance. As Shenhar *et al.* put it, 'how does the current project help prepare the organization for future challenges?' (Shenhar *et al.* 2001).

Employment Opportunities. Some literature on the motivation of FLOSS developers suggests that developers participate to improve their employment opportunities (e.g. Lerner and Tirole 2002).

[5] http://freshmeat.net/faq/view/27/

Thus, one can consider *salary or jobs acquired* through the involvement in a particular project as possible measures of success (Hann *et al*. 2004). 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 measure 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[6] 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 and systems that have incomplete participation.

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, one can consider the 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.

Table 2. Measures suggested by a re-examination of the FLOSS process

Measure of success	Indicators	Audience
Project output	Movement from alpha to beta to stable Achieved identified goals Developer satisfaction	Developers
Process	Number of developers Level of activity (developer and user contributions, number of releases) Time between releases Time to close bugs or implement features	Developers, users
Outcomes for project members	Individual job opportunities and salary Individual reputation Knowledge creation	Developers

2.2.4. *Summary of Measures from Process Reconsideration*

In summary, consideration of the process of developing FLOSS suggests a number of additional measures indicative of success for these projects. These measures are summarized in Table 2. We note that as the measures move further back in the process model, they become increasingly removed from the user. As such, there may be a concern about their validity as measures of success: is it a success if a project attracts developers but not users? Or if it develops high quality processes but not high quality code? We have two replies to this concern. First, the apparent disconnect may be an accurate representation of the reality of FLOSS projects, in which the developers frequently are the users. Second, the measures developed in this section should be viewed as complements to rather than replacements for the more conventional measures of success. Using a variety of measures will provide a richer picture of the status of a project. As well, because many of the individual measures seem likely to have measurement problems, or measure success from different perspectives, adopting a portfolio of measures seems prudent.

[6] http://www.advogato.org/trust-metric.html

2.3. Seeking Input from Floss Developers

Having reviewed the IS literature on IS success and extended this conceptual model by considering the unique features of the FLOSS development and use environments, we continued our theory building by turning to the FLOSS community for input on their definitions of success. Our goal was to extend our conceptual model by generating a range of ideas to compare and extend our list of factors, rather than the relative importance of each one or the relations among them.

To solicit input, we posted a question soliciting feedback on SlashDot[7] a Web-based discussion group that attracts considerable interest and participation from FLOSS developers and users. This data elicitation technique was more like an on-line focus group (or perhaps the initial stage of a Delphi study) than a survey, as respondents were a nonrandom sample, and could see and respond to earlier postings. The rationale for this approach to data elicitation was to match our goal of generating ideas about success measures, rather than testing a theory or making inferences from generalizable data. To elicit comments, the following question was posted on SlashDot on 22 April 2003 (http://slashdot.org/article.pl?sid=03/04/21/239212):

> 'There have been a number of discussions on SlashDot and elsewhere about how good projects work (e.g. Talk To a Successful Free Software Project Leader), but less about how to tell if things are going well in the first place. While this may seem obvious, most traditional definitions of software project success seem inapplicable (e.g. profit) or nearly impossible to measure for most projects (e.g. market share, user satisfaction, organizational impact). In an organizational setting, developers can get feedback from their customers, the marketplace, managers, etc.; if you're Apache, you can look at Netcraft's survey of server usage; but what can the rest do? Is it enough that you're happy with the code? I suspect that the release-early-and-often philosophy plays an important role here. I'm asking not to pick winners and losers (i.e. NOT a ranking of projects), but to understand what developers look at to know when things are going well and when they're not.'

The question received 201 responses within a few days. A transcript of responses was downloaded on 26 April 2003. Many of the individuals posting answers to our question identified themselves as developers or contributors to FLOSS projects. As a check on their qualifications, we searched SourceForge for information about the posters. Although SourceForge and SlashDot are separate sites, many developers have strong attachments to their user IDs and use the same one whenever possible, providing a possible link between the two systems. For example, it seems reasonable to expect that the user ID `Abcd1234` identifies the same individual on both systems. We identified the SlashDot IDs of 72 posters who provided useful responses (some responses were anonymous). Of these seventy-two, 34 IDs matched a SourceForge ID exactly, and six could be matched with a bit of research (e.g. by matching the real name of the individual; real names are available for a few SlashDot posters and many SourceForge developers). Of the matched IDs, 16 and 3 respectively were listed as members of SourceForge projects (i.e. about half). A few other posters had pointers to non-SourceForge FLOSS projects on their SlashDot information page or information about their employment, generally as software developers. These data are not conclusive, but do suggest that a number of the contributors to the study had sufficient background as FLOSS developers to be able to comment knowledgeably.

The transcript was content-analyzed by two coders. The content analysis process was carried out using Atlas-ti, a qualitative data analysis software package. Messages were coded using the thematic unit as the unit of analysis. Once a measure was identified within a message, the coder selected the text containing the measure and coded that text using categories from our coding scheme. A total of 170 thematic units were identified and coded in 91 responses (i.e. some postings contained multiple units; the remaining responses did not contain text addressing the question, e.g. a posting containing an advertisement). The content analysis process employed a mixture of deductive and inductive procedures. The initial content analytic scheme was based on the literature review described above. During the process of content analysis, additional themes emerged from the data. Data analysis

[7] http://slashdot.org/

Table 3. Results of the content analysis of SlashDot responses

Level 1	Level 2	Frequency	Percentage (%)
User	Satisfaction	14	8
	Involvement	25	15
Product	Meeting requirements	9	5
	Code quality	11	6
	Portability	1	1
	Availability	2	1
Process	Activity	5	3
	Adherence to process	10	6
	Bug fixing	4	2
	Time	2	1
	Age	1	1
Developers	Involvement	16	9
	Varied developers	2	1
	Satisfaction	29	17
	Enjoyment	8	5
Use	Competition	4	2
	Number of users	2	1
	Downloads	3	2
Recognition	Referral	3	2
	Attention and recognition	9	5
	Spin-offs	6	4
Influence		4	2
	Total	170	

continued until saturation was reached. The two raters agreed on the codes for 78% of the units. We felt that this level of agreement was sufficient for the purposes of the analysis (identification of measures to compare to the literature review), so we did not go on to refine the definitions of codes or retrain the coders to increase agreement. The results of the content analysis are summarized in Table 3.

The codes were organized into a two-level hierarchy for presentation, with detailed codes (Level 2 in the table) clustered into meta-categories (Level 1). 32% of the units included elements from the Developers meta-category, indicating that the respondents felt that a project is successful if the developers are involved, satisfied, enjoyed the process and that there is a variety of them. The Users meta-category also had a large number of responses. 23% of units indicated that the poster felt a project was successful if it satisfies users (other than developers) and that users are involved in discussions and bug reports. Involvement of both users and developers was frequently mentioned, accounting for 31% of the units. Project recognition codes were found in 11% of the units, exceeding the number of responses indicating Use as a measure of success, which accounted for 5% of instances. Finally, the product's Quality (13%) and Process (13%) were suggested to be measures of success by developers as well. Note that percentages are reported only to more completely describe the data. Given the nonrandom sample of contributors and the open data elicitation technique, the frequency of a response should not be interpreted as important.

Overall, the responses of the developers posting on SlashDot were in general agreement with the list of success measures we developed from the literature and our re-examination of the process. The analysis indicates that developers found their personal involvement, satisfaction and enjoyment to be measures of the success of a project, consistent with the view of FLOSS as 'software that scratches an itch'. However, some new themes did emerge from the coding.

- First, a number of respondents suggested recognition (e.g. mention on other sites) as a measure of project success. This measure could be operationalized by searching the Web for the project name (for projects with unique names) or for the URL of the project's home page. A related suggested measure was the influence of the product or project's process on other FLOSS groups and other commercial settings. These responses are consistent with the literature on FLOSS developers' motivations that suggest recognition as a primary motivation for involvement.
- A second category that emerged was 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. We had considered contributions from developers, but these responses emphasize the fact that FLOSS projects are also dependent on help from users to identify problems and post suggestions.
- A final category that emerged from the data was the issue of porting, that is, moving a system from one operating system platform to another. 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.

What was also surprising was what respondents did not say, in that none of the respondents mentioned some of the measures of success we had identified. For example, though several authors have suggested that developers are motivated by the chance to learn and perhaps get a better job, none of the respondents mentioned these factors. A possible explanation is a strong community norm on SlashDot that endorses altruism over expressions of self-interest, which may have restricted discussion in that non-anonymous and community-moderated forum.

2.4. Success Measures in Recent Floss Research

In the previous sections, we developed a list of possible success measures for FLOSS projects on the basis of a review of the IS literature, with extensions based on a consideration of the FLOSS development process and feedback from FLOSS developers. In this section, we examine the current state of academic research on FLOSS. To understand how success has been operationalized in recent FLOSS research, we carried out a content analysis of the working articles posted on the http://opensource.mit.edu/pre-press Web site. As of 1 March 2005, the site included 182 recent working articles and abstracts, thus presenting a convenient cross section of the literature. The rationale for using this sample of articles was that the abstracts for the articles were all on one page, making it much easier to collect them (as against doing manual searches across multiple journal Web sites), the articles were recent, having not been delayed by the publication cycle, and the articles presented a broad cross section of FLOSS research, while journal publications likely have some selection based on the interests of the journal.

Four people coded the articles on the site for type of article (empirical *vs* conceptual) and for use of project success as a dependent variable (if any). The articles were divided into three groups for initial coding by three of the coders, and the empirical articles were then recoded by a fourth coder to ensure comparability in the coding. Of the 182 articles, 84 were deemed to be empirical studies. Of these only 14 were identified as studying success, performance or effectiveness of project teams in some manner. Table 4 shows the specific concepts and measures we found with citations to the associated articles.

There are two striking features of this table. Firstly, there is a wide range of alternative measures of project success; the field has not settled on any one measure, or systematic group of measures. Secondly, there are a great many measures related to the process of system creation itself, especially if one includes 'learning by developers' as an outcome measure internal to the team. This result is pleasingly consistent with reconsideration of IS success models in the FLOSS context presented above, which pointed to the early stages of systems development as particularly important to an understanding of IS success in the FLOSS context.

3. SUMMARY OF CONCEPTUAL SCHEME FOR FLOSS SUCCESS

Table 5 presents a summary of the success measures, and possible operationalizations that we have discussed above. Using our reconsideration of the FLOSS process as its framework, the table draws together the insights from the review of the IS literature, our extension of these models in light of the FLOSS process, input from FLOSS developers via SlashDot and our analysis of the existing literature studying FLOSS.

4. EMPIRICAL STUDY OF SUCCESS MEASURES USING SOURCEFORGE DATA

The previous section of this article developed a conceptual model of success factors on the basis of the literature and theoretical considerations and compared these to developer opinions and the current state of the empirical FLOSS research. In this section, we continue our examination of success measures empirically, by using data from SourceForge. This study demonstrates the operationalization of three of the measures we suggest above and allows us to assess their convergent validity. Following our report of this study, we consider the implications of our findings for our theory building and to suggest future avenues of research.

From the measures developed above, we chose the number of developers (assessed from the records of the project and from bug-fixing logs), bug-fixing time and popularity (assessed from the number

Table 4. Success measures in recent FLOSS research

Type	Measure	Operationalization	Example citations
System creation	Activity/effort	SourceForge activity level	Crowston *et al*. 2004 Crowston and Scozzi 2002
		Time invested per week by administrators	Stewart and Gosain in press
		Time spent in discussion	Butler *et al*. 2002
		Number of message posts on SourceForge	Krishnamurthy 2002
	Size of development team	Number registered as developers on SourceForge	Krishnamurthy 2002, Stewart and Gosain in press
		Number checking code into CVS	Mockus *et al*. 2000, Reis and Fortes 2002
		Number of posters on mailing lists	Mockus *et al*. 2000
		Number of posters in bug tracker	Crowston *et al*. 2004
	Programmer productivity	Lines of code per programmer per year	Mockus *et al*. 2000
	Project development status	Active or not on SourceForge	Giuri *et al*. 2004
		Development stage on SourceForge	Crowston and Scozzi 2002 Krishnamurthy 2002, Stewart and Gosain in press
	Task completion	Speed in closing bugs or tracker items	Mockus *et al*. 2000, Stewart and Gosain in press Crowston *et al*. 2004
	Development of stable processes	Description of development processes	Reis and Fortes 2002
System quality	Modularity	Modularity of source code	Shaikh and Cornford 2003
	Correctness	Defects per lines of code/deltas	Mockus *et al*. 2000
	Manageability	Lines of code under package management	González-Barahona and Robles 2003
	Maintainability	Common coupling	Schach *et al*. 2005 Schach *et al*. 2002
System use	Interest	SourceForge page views	Krishnamurthy 2002, Stewart and Gosain in press Crowston *et al*. 2004
	Copies in circulation	SourceForge downloads	Krishnamurthy 2002, Stewart and Gosain in press Crowston *et al*. 2004
	Market share	Netcraft survey	Mockus *et al*. 2000
	Support effectiveness	Number of questions effectively answered	Lakhani and Wolf 2003
		Time spent seeking help/providing help	Lakhani and Wolf 2003
System consequences	Learning by developers	Motivations of developers for reading and answering questions	Lakhani and Wolf 2003
	Tool development	Description of tools	Reis and Fortes 2002

of downloads and viewings of project Web pages), and inclusion in distributions. These measures were chosen because they span the reconsidered FLOSS development process discussed above, including inputs (number of developers), process (speed of bug fixing) and output (popularity), and because the data were representative of the kinds of data used in prior research.

Our analysis aims at assessing the utility of these measures for future research. Each has good face validity, in the sense that a project that attracts developers fixes bugs quickly, and which is popular does seem like it deserves to be described as a success. We are also interested in assessing how these measures relate to one another: do they measure the same construct or are they measuring

Table 5. Summary of concepts for Information Systems success in FLOSS context

Process phase	Measure	Potential indicators
System creation and maintenance	Activity/effort	File releases, CVS check-ins, mailing list discussions, tracker discussions, surveys of time invested
	Attraction and retention of developers (developer satisfaction)	Size, growth and tenure of development team through examination of registration, CVS logs. Posts to dev. mailing lists and trackers. Skill coverage of development team. Surveys of satisfaction and enjoyment
	Advancement of project status	Release numbers or alpha, beta, mature self-assessment, request for enhancements implemented
	Task completion	Time to fix bugs, implementing requests, meeting requirements (e.g. J2EE specification). Time between releases
	Programmer productivity	Lines of code per programmer, surveys of programmer effort
	Development of stable processes and their adoption	Documentation and discussion of processes, rendering of processes into collaborative tools, naming of processes, adoption by other projects/endeavors
System quality	Code quality	Code analysis metrics from software engineering (modularity, correctness, coupling, complexity)
	Manageability	Time to productivity of new developers, amount of code abandonment
	Documentation quality	Use of documentation, user studies and surveys
System use	User Satisfaction	User ratings, opinions on mailing lists, user surveys
	Number of users	Surveys (e.g. Debian popularity contest), downloads, inclusion in distributions, package dependencies, reuse of code
	Interest	Site pageviews, porting of code to other platforms, development of competing products or spin-offs
	Support effectiveness	Number of questions effectively answered, time required to assist newbies
System consequences	Economic implications	Implementation studies, e.g. total cost of ownership, case studies of enablement
	Knowledge creation	Documentation of processes, creation of tools
	Learning by developers	Surveys and learning episode studies
	Future income and opportunities for participants	Longitudinal surveys
	Removal of competitors	Open sourcing (or substantial feature improvement) of competing proprietary applications

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?

4.1. Method

In order to assess the utility of these measures, we first developed operationalizations for each measure on the basis of data available on the SourceForge Web site. We then collected data from the SourceForge system, using Web spiders to download the html pages and parsing out the relevant fields. With the data in hand, we extracted each individual measure, achieving results, which we present below. We then examined the measures' relationships using a correlation matrix (with Bonferonni corrections for multiple correlations) to see if these measures measure the same or different things.

4.1.1. Operationalization

Inputs: Developer Counts. We operationalized the number of developers involved in a project in two ways. First, we extracted the developer count from the SourceForge project summary pages. Alternative operationalizations considered included analyzing the CVS logs to count the developers contributing, but this method assumes that all projects allow all developers direct access to the CVS (rather than using a patch submission system) and, more problematically, that only those contributing code to CVS should be counted as developers. Therefore, we used the project's own definitions of developers. The counts on the SourceForge summary page are self-reported data, but since being listed as a developer there involves both an application and approval by a project administrator, it provides a useful measure of developer involvement. In order to assess how well projects were doing in attracting and retaining developers, we analyzed the change in developer counts over time. A project that has a growing number of developers is more successful in attracting developers than one that has a shrinking number. These changes were analyzed both as categorized time series and using a weighted delta more appropriate for our intended correlation study.

Second, since the FLOSS development process relies on contributions from active users as well as core developers, we developed a measure that reflected the size of this extended team, rather than just the core developers. As a proxy for the size of the extended development community, we counted the number of individuals who posted a bug report or message to the SourceForge bug tracker. Alternative operationalizations of this construct would include counting posters on the various mailing lists, including development lists and user-support lists. Analyzing instead the bug tracker was practically convenient (as we were already collecting that data), but participation there also demonstrates closer involvement in the project than just posting user questions to the mailing list, as well as being a venue where direct interaction between users and developers would be found.

Process: Speed of Bug Fixing. We operationalized team performance in the speed of bug fixing by turning to the bug tracker provided to SourceForge projects. We examined how long it took the program to fix bugs by calculating the lifespan of each bug from report to close using the opened and closed timestamps recorded by the bug tracker. 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 (most bugs are closed quickly, but a small number take much longer), 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' data. Simply leaving out these unclosed bugs would bias the estimated time to fix bugs. 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. In Section 4.3 we describe how we approached these difficulties using the statistical approach known as *survival* or *event history analysis*.

Outputs: Popularity. Our final measure, popularity, was assessed in three ways. First, we extracted the number of *downloads* and project *page views* reported on the SourceForge project pages.[8] Because some projects were older than others, we operationalized this aspect of popularity as downloads and page views per day. In keeping with a portfolio approach to success measurement, we also measured popularity by examining whether the project produced programs that were *included in the Debian Linux distribution*, the largest distribution of FLOSS. Debian is distributed as a base installation and a set of additional packages for different programs. Not all the programs produced by FLOSS projects are candidates for inclusion in a Linux distribution (for example, some projects are written only for the Windows or Mac OS X platform), so this measurement was taken only for projects producing programs eligible for inclusion in the distribution.

4.2. Data Collection

To gather data about the projects, we developed a spider to download and parse SourceForge project pages. Spidering was necessary because the SourceForge databases were not publicly available.[9] Data

[8] The SourceForge Web site at the time of data collection noted that 'Download statistics shown on this page for recent dates may be inaccurate', but our examination of the data suggests a systematic under-reporting, rather than a bias in favor of one project or another. As a result, while the absolute numbers may be wrong, the data are sufficient to indicate relative performance of projects and the relationships of these data to other variables.

[9] In the period between our research and this publication a limited amount of SourceForge data became available directly from database dumps provided to researchers via a group based at Notre Dame (http://www.nd.edu/~oss/Data/data.html). It is our understanding that this data source does not include tracker or mailing list information, but certainly if the conditions are acceptable to researchers and the data adequate, this appears to be an excellent source of clean data.

were collected at multiple points to allow for a longitudinal analysis. We collected data in February 2001 and April 2002. We also obtained data from Chawla *et al*. for October 2003 (Chawla *et al*. 2003) and from Megan Conklin for October 2004 and February 2005 (Conklin 2004). These data have been collated and made available via the FLOSSmole project (http://ossmole.sourceforge.net/, Howison *et al*. In press). The use of data from multiple points in time provides a dynamic view of the projects lacking in most analyses of FLOSS projects.

At the time we started our study, SourceForge supported more than 50,000 FLOSS projects on a wide diversity of topics (the number was 78,003 as of 21 March 2004 and 96,397 projects and more than 1 million registered users by 28 February 2005). Clearly not all of these projects would be suitable for our study: many are inactive, many are in fact individual projects rather than the distributed team efforts we are studying, as previous studies have suggested (Krishnamurthy 2002), and some do not make bug reports available. While we were able to assess these difficulties at the time of our original project selection, the FLOSSmole data spans five years and covers all the projects on the site over that period and so we illustrate our next point with that more complete data. With respect to our first measure, developers listed on SourceForge homepages, our data confirmed the impression gained from previous research that few SourceForge projects represent team efforts. Of the 98,502 projects that we examined in the SourceForge system up to February 2005, 64,881 (67%) never had more than one developer registered to the project at any time in the five years, as shown by the distribution of developer counts shown in Figure 3.

It was also clear that not all projects used the bug tracking system sufficiently to allow the calculation of community size, nor the speed of bug fixing. In order to collect useful data for these measures we restricted our study to projects that listed more than seven developers and had more than 100 bugs in the project bug tracker at the time of selection in April 2002. This restriction was justified theoretically as well as practically: having multiple developers suggests that the project is in fact a team effort. Having bug reports was a necessary prerequisite for the planned analysis, as well as indicative of a certain level of development

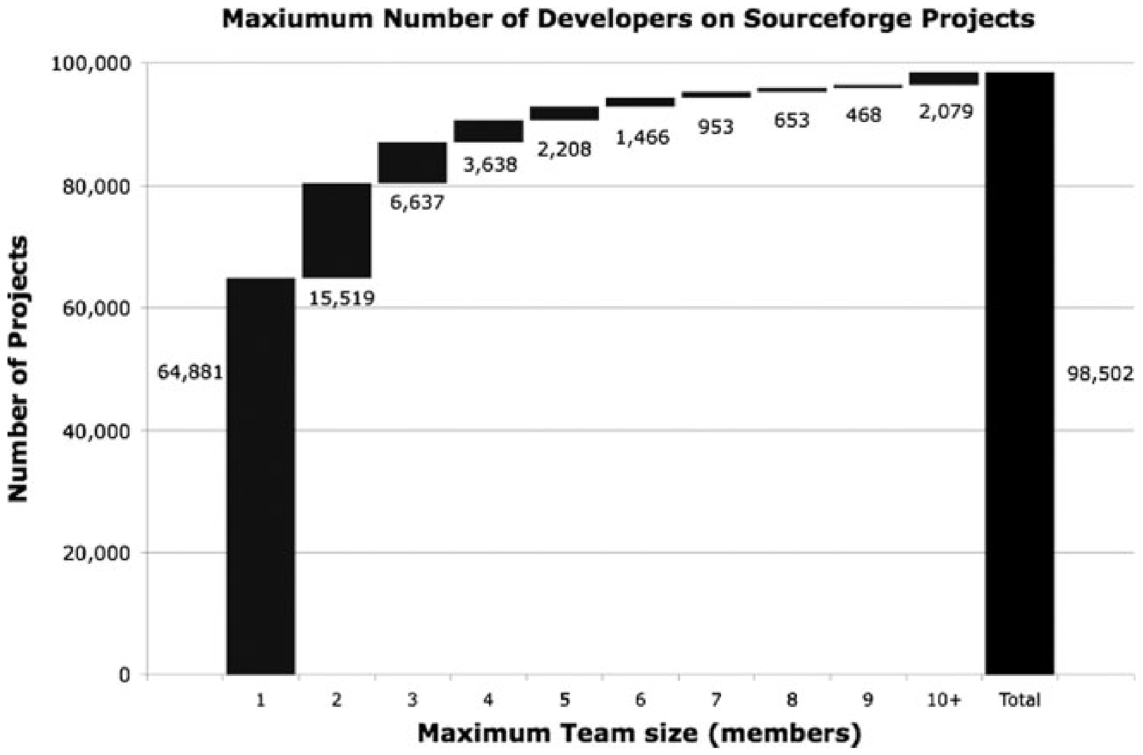
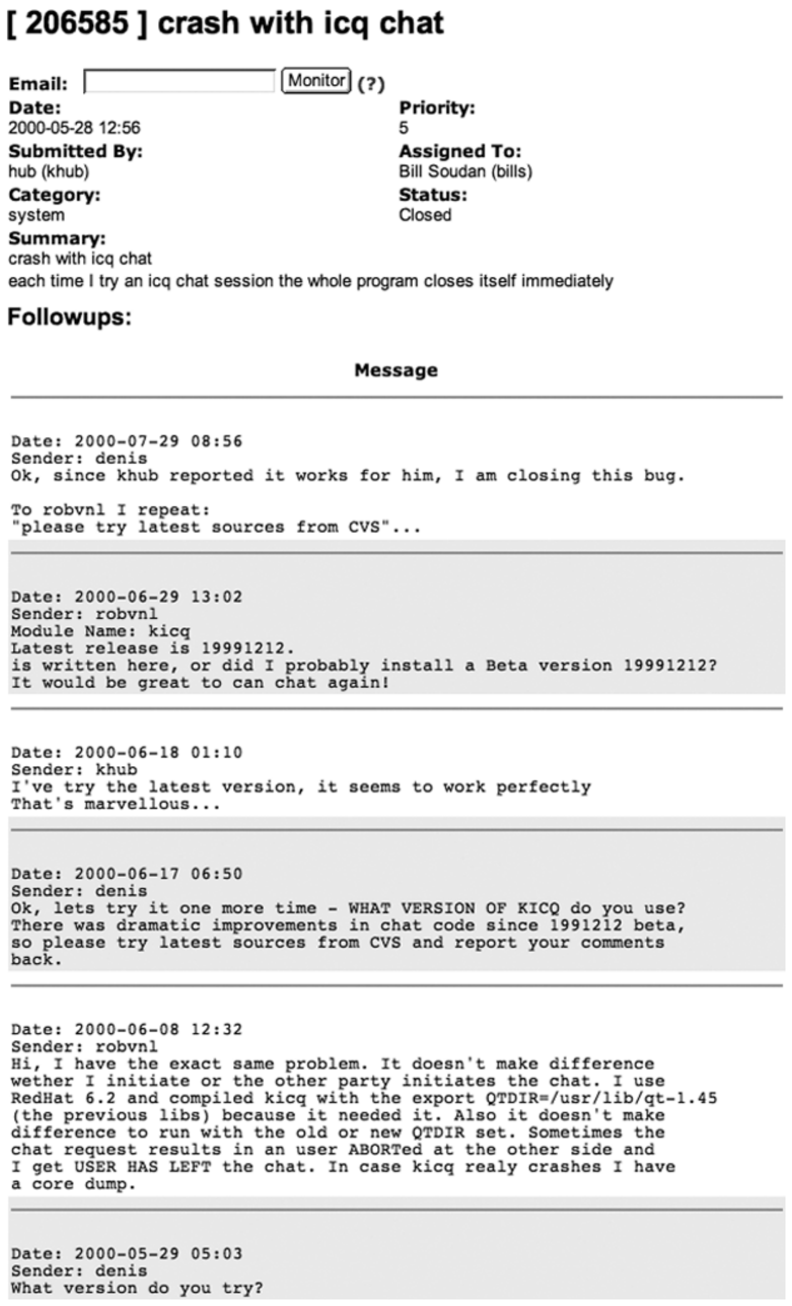


Figure 3. Maximum number of listed developers per project on SourceForge projects over the 5-year period from 2001 to 2005



[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?
```

Figure 4. Example bug report and followup[10] messages

effort. Quite surprisingly, we identified only 140 projects that met both criteria. The sample includes the projects `curl`, `fink`, `gaim`, `gimp-print`, `htdig`, `jedit`, `lesstif`, `netatalk`, `phpmyadmin`, `openrpg`, `squirrelmail` and `tcl`. Those familiar with FLOSS will recognize some of these projects, which span a wide range of topics and programming languages.

To study the performance of bug fixing, we collected data from the SourceForge bug tracking system, which enables users to report bugs and to discuss them with developers. As shown in Figure 4, 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,

[10] Adapted from http://SourceForge.net/tracker/index.php?func=detail&aid=206585&group_id=332&atid=100332

we developed a spider program that downloaded and parsed all bug report pages for the selected projects. The spider was run in March 2003. 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.

Once obtained and parsed we conducted a basic exploration of the data for the purposes of data cleaning, which revealed problems with the quality of the data for some of the projects. For example, one team had apparently consolidated bug reports from another bug tracking system into the SourceForge tracker. These copied-over bugs all appeared in SourceForge to have been opened and closed within minutes, so this project was eliminated from further analysis. Another project was eliminated because all of the bug reports were in Russian, making the data impossible for us to interpret (and apparently for others as well: only three posters had participated, despite nine developers being registered to the project). As a result, the sample for the remainder of the analysis was reduced to 120 projects. We also deleted a few bug reports where the computed lifespan was 0 seconds or less due to errors in the data or in the parsing. We obtained data on a total of 56,641 bug reports, an average of 472 per project. The median number of reports was 274, indicating a skewed distribution of number of bug report per project.

4.3. Analysis

In this section we present the analyses we performed for each of our measures, deferring discussion of the results to the next section.

Popularity. Of our two measures of popularity, number of developers and community size, community size required little detailed analysis. We simply counted the unique number of posters to the bug trackers on SourceForge, dropping only the system's indicator for anonymous posting.

The analysis of developer numbers was more involved because, as mentioned above, a unique feature of our data on developer numbers is that we had data over time, which allowed us to assess how the number of developers in a project changed over time, and we wanted to take advantage of that. However, since we had only five data points, measured at inconsistent time intervals, it was impossible to apply standard time series analysis techniques. We therefore simply grouped projects into six categories on the basis of how the number of developers had changed over time – whether it had risen, fallen or stayed the same, and how consistently it had done so.[11] The precise definitions of the six categories is presented in Table 6. Figure 5 shows the series, allowing a visual inspection to assess the reasonableness of the categorizations.

A visual inspection of the diagrams suggests that the grouping of the patterns is reasonable. Figure 5(a) shows the trend for the high number of projects in our sample that have continually attracted developers. Figure 5(b) shows the trend for projects that tend to grow but which have, at times, fallen. Figure 5(c) shows unchanged projects, or projects that move up and down without a trend. Not all the projects in this category were started at the time of our first data collection, so further data collection may reveal a trend and facilitate sorting into one of the other categories. Figure 5(d) shows projects that have fallen for at least half of their life spans and risen in only one period. Interestingly the rise is almost always early in the life of the project, followed by a sustained decline. These projects appear to have attracted some initial interest but were unsuccessful in retaining developers. This may indicate troubles in the teams despite an interesting task. Figure 5(e)

Table 6. Definitions of categories of pattern of change in developer counts

Category	Description
1. Consistent risers	No falls, rising: a pattern of consecutive rises at least half as long as the whole series was found[a]
2. Risers	Mostly rising but with at least one fall
3. Steady or not treading	Unchanged or neither rising nor falling
4. Fallers	At most one rise but mostly falling: a pattern of consecutive falls at least half as long as the whole series was found[a]
5. Consistent fallers	Always falling, no rises at all
6. Dead projects	Project removed from SourceForge

[a] A rise (or fall) followed by no change was counted as two consecutive rises (or falls).

[11] We have made the code for this categorization available via the FLOSSmole project

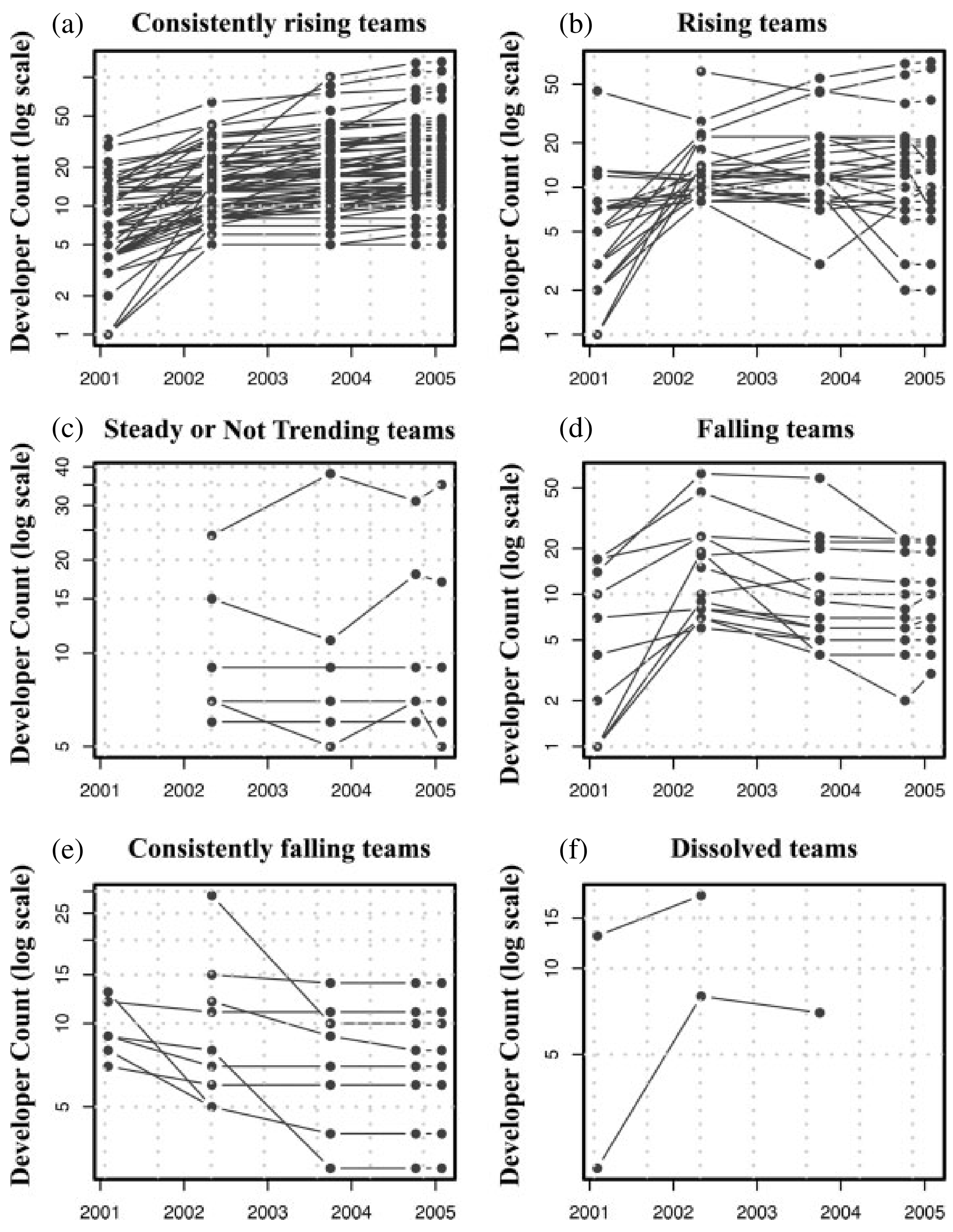


Figure 5. Changes in developer counts per project over time, categorized by trend. (Log scale: note that the vertical axis is inconsistent with lower total numbers in (e) and (f))

shows projects that have never grown and have lost members for at least two consecutive periods. Finally, Figure 5(f) shows projects for which data became unavailable. These are dissolved projects that have been removed from SourceForge. The results of comparing these categorizations with the whole SourceForge population are presented below in Section 4.5, which allow us to interpret our results in the full context.

While the developer series categorization was useful, it was not suitable for the correlation study of the proposed success measures. For that purpose, we computed a weighted average of the changes from period to period, using as weights the inverse of the age (1/1 for the change from 2004 to 2005, 1/2 for the change from 2003 to 2004, etc.). This procedure was adopted on the basis of the theory that the ability to attract and retain developers is an

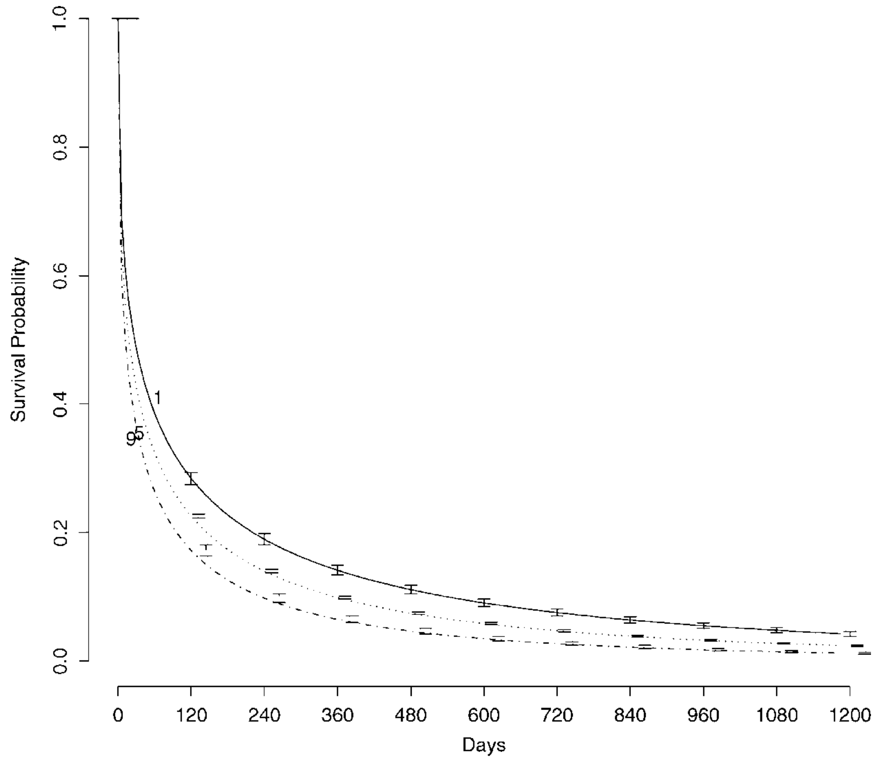


Figure 6. Plot of bug survival *versus* time for high (9), default (5) and low (1) priority bugs

indicator of success and to focus attention on the growth (or decline) in developers and to give more weight to recent experience. This weighted average figure is reported below:

Community Size. 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. We counted a total of 14,922 unique IDs, of whom 1280 were involved in more than one project (one was involved in eight of the projects in our sample). The total counts per project were log-transformed to correct skew.

Bug-fixing Time. As discussed above, the analysis of bug-fixing speed is complicated by the right-censored data available. 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 a hazard function, which is the instantaneous probability of a bug being fixed at any point during its life or equivalently the survival function, which is the percentage of bugs remaining open. A plot of the survival over time for all bugs in our sample is shown in Figure 6. The plot shows that bugs with higher priorities are generally fixed more quickly, as expected, but some bugs remain unfixed even after years.

The hazard function (or more usually the log of the hazard function) can be used as a dependent variable in a 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 for the dummy variable for each project in the hazard function, 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. We experimented with different functional forms for fitting the bug hazard rates. Somewhat surprisingly, the form that fitted best was an exponential (the R^2 for the fit was 0.51), that is, one in which the hazard rate is not time varying. Therefore the hazard ratio for each project is reported below:

Popularity. The counts of project downloads and page views extracted from the downloaded html pages were divided by age to compute downloads and page view per day and log-transformed to correct skew.

Our additional measure of popularity, inclusion in a distribution, required additional analysis. We first examined the list of Debian packages manually to match packages to the projects in our SourceForge sample. To do this for each project we examined the output of the Debian `apt-cache` program, which searches package names and descriptions, and the http://packages.debian.org/site, which allows searching filenames within package contents. We then examined the SourceForge homepages of the projects to be sure that we had an accurate match. For those that were included we observed that one project was linked to one or more (sometimes many more) packages, but we did not observe many projects that were packaged together. As discussed above, some of the projects in our sample are not candidates for inclusion in the Debian distribution because they do not run on Linux and were not included in this measure. For example, `fink` is a package-management system for Mac OS X and `Gnucleous` is a Windows client for the Gnutella network. We found that there were 110 packages (out of 120, or 92%) that could be installed and run on a Debian Linux system and 63 (57%) of these were packaged and distributed by Debian, while 47 (43%) were not. One package was undergoing the Debian quality assurance process but was not yet available through the standard installation system and was therefore coded as not included. Projects with eligible programs that were included in Debian were given a score of 1 (or YES), and projects without included programs were given a score of 0 (or NO); ineligible projects were tagged with NA.

4.4. Results

The results of measuring these success measures across our sample of 120 projects are presented below. First we present descriptive statistics for the individual measures and then present the results of examining the correlations between the measures. We conclude our results discussion by reporting on the practicality of our operationalizations. Table 7 shows the descriptive statistics for the individual measures.

Table 7. Descriptive statistics for sample success measures

Variable	Mean	Median	SD
Lifespan (days)[a]	1673	1699	198
Developers in 2001	8.63	7	7.20
Developers in 2002	15.56	12	11.22
Developers in 2003	18.19	12	16.29
Developers in 2004	20.06	14	19.95
Developers in 2005	20.22	14	20.57
Weighted delta (see text)[a]	1.60	0.72	3.26
Posters to bug tracker[a]	140	86	207
Log bugs[a]	5.78	5.61	0.84
Closed bugs[a]	405	234	489
Log median bug lifetime[a]	14.44	14.39	1.29
Hazard ratio[a]	1.13	1.10	1.04
Log downloads (all time)[b]	11.29	11.87	3.38
Log downloads (per day)[b]	4.32	4.44	2.24
Log page views (all time)[b]	13.85	14.15	2.14
Log page views (per day)[b]	6.45	6.74	2.12
Debian package?	63 Yes	47 No	10 NA

[a] $N = 120$.
[b] $N = 118$.

To examine the relationships among the variables we measured, we examined the correlations, given in Tables 8 and 9. Forty-two of the 136 correlations are statistically significant indicating that there is a genuine relationship. (With 120 cases and applying the Bonferonni correction for the 136 comparisons possible among 17 variables, the critical value for significance at $p = 0.05$ is $r = 0.32$.) None of the proposed success measures are correlated with project lifespan, suggesting that they do provide some indication of the performance of the project rather than just accumulation of events.

Bold and underlined correlations are significant at $p < 0.05$, using a Bonferonni correction for the number of correlations.

The counts of number of developers at different points in time are correlated (the upper box in Table 8), as would be expected given that they constitute a time series. The counts are also correlated with the computed weighted average of changes in developers. Interestingly, the developer counts are also correlated to number of bugs reported (the lower box in Table 8). It may be that developers themselves post bug reports and so more developers constitutes more activity. Alternately, it may be that activity attracts developers. As well, the count of participants in the bug tracker is correlated with number of bugs, but not with number of listed developers (the upper left box in Table 9). These relationships suggest that individuals post only

Table 8. Correlations among sample success measures

	Lifespan	Developers in 2001	Developers in 2002	Developers in 2003	Developers in 2004	Developers in 2005	Weighted delta
Lifespan	1.000						
Developers in 2001	0.265	1.000					
Developers in 2002	0.061	**0.627**	1.000				
Developers in 2003	−0.016	**0.648**	**0.761**	1.000			
Developers in 2004	−0.047	**0.656**	**0.643**	**0.949**	1.000		
Developers in 2005	−0.042	**0.658**	**0.643**	**0.943**	**0.998**	1.000	
Weighted delta	−0.023	**0.374**	**0.352**	**0.789**	**0.913**	**0.922**	1.000
Posters to bug tracker	0.052	0.144	0.266	0.200	0.226	0.230	0.186
Log bugs	0.008	0.158	**0.396**	**0.392**	**0.398**	**0.394**	0.316
Closed bugs	0.034	0.191	**0.381**	**0.322**	**0.344**	**0.348**	0.272
Log median bug lifetime	0.087	0.166	0.103	0.208	0.190	0.184	0.172
Hazard ratio	0.200	0.219	0.084	0.177	0.158	0.147	0.134
Log downloads (all time)	0.136	0.056	0.198	0.212	0.241	0.238	0.240
Log downloads (per day)	0.010	0.059	0.228	0.249	0.295	0.287	0.283
Log page views (all time)	0.001	0.043	0.228	0.245	0.277	0.279	0.269
Log page views (per day)	−0.060	0.035	0.224	0.245	0.280	0.282	0.271
Debian package?	0.285	0.153	0.116	0.101	0.064	0.054	0.054

Table 9. Correlations among sample success measures, continued

	Posters to bug tracker	Log bugs	Closed bugs	Log median bug lifetime	Hazard ratio	Log downloads (all time)	Log downloads (per day)	Log page views (all time)	Log page views (per day)
Posters to bug tracker	1.000								
Log bugs	**0.592**	1.000							
Closed bugs	**0.801**	**0.718**	1.000						
Log median bug lifetime	0.179	0.087	0.232	1.000					
Hazard ratio	0.158	0.114	0.239	**0.868**	1.000				
Log downloads (all time)	**0.359**	0.268	0.252	0.095	0.068	1.000			
Log downloads (per day)	**0.450**	**0.343**	0.312	0.088	0.017	**0.909**	1.000		
Log page views (all time)	**0.435**	**0.462**	**0.355**	0.031	0.146	**0.660**	**0.724**	1.000	
Log page views (per day)	**0.433**	**0.463**	**0.354**	0.038	0.160	**0.647**	**0.722**	**0.998**	1.000
Debian package?	0.125	0.073	0.088	0.063	0.115	0.201	0.240	0.171	0.150

a few bug reports, so more bug reports implies a greater number of participants. The correlation between posters and the number of closed bugs is particularly strong ($r = 0.718$).

The counts of downloads and page views (all time and daily) are all strongly correlated (the lower right box in Table 9), suggesting that they offer similar measures of popularity. They are also correlated with the number of bugs (the lower left box in Table 9) and the count of participants in the bug tracker. These correlations taken together suggest that the count of participants and number of bugs function more like indications of the popularity of a FLOSS project, rather than the success of its development processes. On the other hand, the hazard ratio for bug lifetimes and the median bug lifetime are not significantly correlated with any of the other variables, suggesting that they do provide an independent view of a project's performance.

4.5. Discussion

The study provided useful data for reflecting on the conceptual model for success in FLOSS development developed in the first half of this article. Because many of these findings reveal themselves as limitations in our study, we first discuss these and attempt to distill general advice for FLOSS research using success as a variable.

The measures applied in this article have good face validity as indicators. The analysis presented above allows us to extend our examination of the validity and utility of the measures. The high correlation among many of these measures indicates a degree of convergent validity, since the different measures do correlate, particularly number of developers and popularity. Examining these correlations and correlations with other variables in more detail suggests room for improvement in the measures. Clearly additional data could be collected to increase the accuracy of the developer counts. Furthermore, our counts are merely aggregate measures that may mask many developers leaving and joining a project. These more specific counts would enable us to determine if some projects are subject to significant 'churn' of individual developers, or conversely of the 'tenure' of individuals as developers on projects. Such a measure might be a theoretically more valuable, as it would have implications for development and retention of knowledge. As with our analysis of bug lifetime, such analyses would need to employ event history statistics to account for the right-censored data.

Further possibilities for measuring developer participation exist, which may provide more accurate and valuable measures. Such opportunities include measuring participation in mailing lists and developer's involvement in code writing directly by examining logs from the CVS system. Each of these measures, however, suffers from the difficulty of inferring departure and 'tenure' because it is possible to 'lurk' in each of these fora. For example, if a developer is observed to post once in 2003 and again in 2005, was the developer always a part of the project, or did the developer leave and rejoin? It might be worth developing a measure of consistent engagement, but it would need to account for different patterns for different individuals.

Another instructive limitation is that in building a sample of projects to study, we seem to have found projects that seem to be mostly successful, by and large. This can be seen in a comparison of the categorization of developer count series from our sample and from the whole SourceForge population. We divided the population of SourceForge projects into the same six categories. Figure 7 shows a comparison of the distribution of time series into our categories between our sample and the total SourceForge population. We required three periods of data collection to calculate the category, so the full population includes only 39,414 projects, excluding 59,154 projects that are either too new or already dead before our fourth measurement point. The comparison shows that our sample of 120 projects has a higher proportion of constant

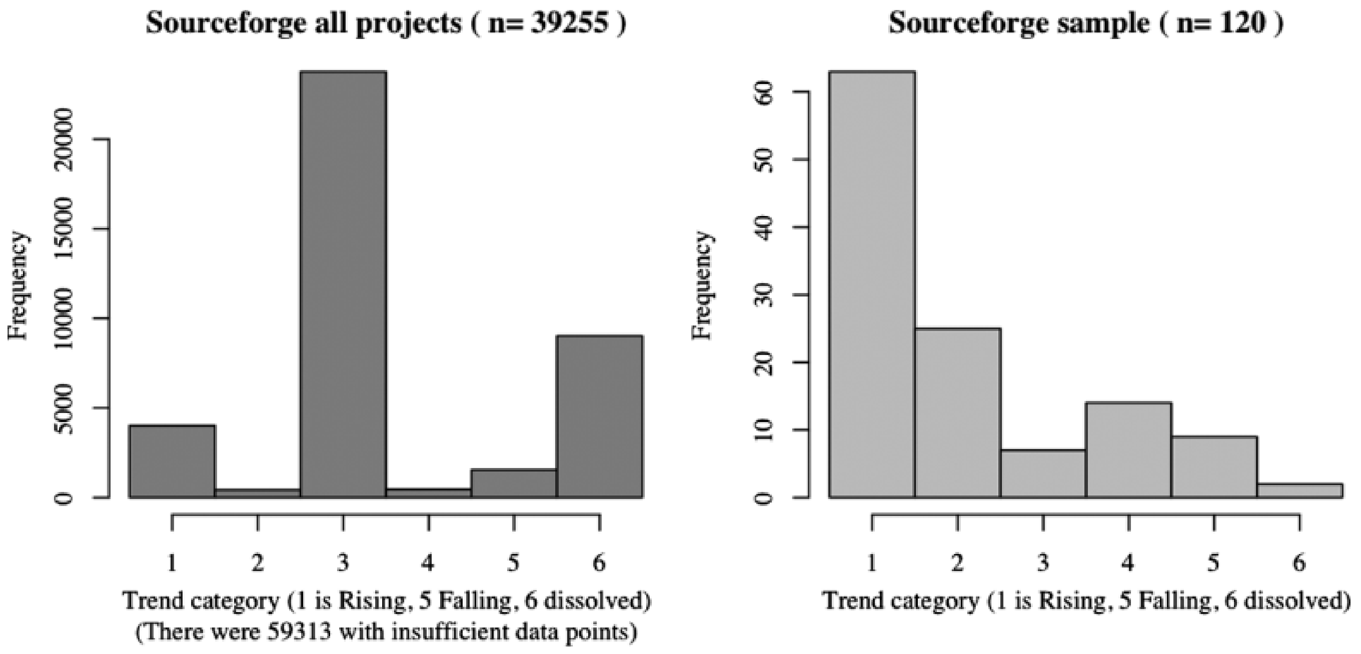


Figure 7. Changes in developer counts per project over time, categorized by trend

risers and a lower proportion of fallers and dead projects. The difference in the distributions is highly significant ($\chi^2 = 805$, $df = 5$, $p < 0.001$). This result suggests that our sample, projects with at least seven developers and 100 bugs in April 2002, is comparatively successful, at least in terms of attracting and retaining developers over time.

As a result of this unintentional bias in our sample construction, the sample may not have sufficient variance on success, affecting the observed correlations reported above. To address this concern, FLOSS research should be aware that selecting on the basis of team size and process features, such as use of the SourceForge trackers, risks selecting only successful projects and should therefore make a special effort to collect data on a broader range of projects, including some that seem clearly to be unsuccessful. This advice is true for quantitative research but is also relevant to case study research; there is a real need for detailed research on failed FLOSS projects.

A second instructive limitation is that we followed the standard but inadequate practice of using popularity measures unadjusted for potential 'market size'. A project's downloads are capped by their total potential downloads, their 'potential market'. A consumer-oriented application, such as an instant messaging client, is of potential usefulness to almost all internet users, whereas a program to model subatomic particle collision (for example) has a much lower potential market. While the instant messaging program might achieve only 40% of its potential market, its absolute number will be far higher than the number achieved by the particle collision modeling tool, even if it is used and adored by 95% of high energy physicists. Thus download figures without market share data are useful only if one considers all programs to be in competition with each other, i.e. if one ignores the project's aims entirely. Some value can be salvaged by using relative growth figures instead of the absolute numbers. Within limited domains of projects, it might be possible to create categories of truly competing products for which the absolute and relative download numbers ought to be a usable proxy for software use. Unfortunately the self-classification of products on sites like SourceForge (where 'End-user software' is an entire category) is of little use and identification of competitors must be carried out by hand.

Furthermore, our analysis revealed that measures such as community size (in numbers of posters) are more similar to these popularity measures than to the process measures. On reflection, community size should also be expected to be heavily influenced by the potential size of the user population, much more so than the smaller developer numbers. A brief inspection of our data on bug posters affirms this: the projects with the largest number of posters are consumer desktop applications, such as `gaim`. Community size measures, therefore, should be adjusted in the same way as downloads and pageviews: by either using within-project changes or by manually creating categories for competing projects.

This discussion draws attention to an element missing from our theory development above and from FLOSS research in general. The phenomenon under research, FLOSS and its development, is generally, often implicitly, defined as any projects that use an 'open source license' (usually defined as OSI approved licenses). Our selection of projects from SourceForge also implicitly uses this definition of the phenomenon, as we studied SourceForge projects, and SourceForge allows only projects using these licenses to register with the site. The risk here is that there are a wide range of system types, developer goals, software development processes, management styles and group structures all of which are compatible with using an OSI approved license. It is not yet clear upon what characteristics the phenomenon should be divided, but it is clear that the meaning of success for different types of projects, programs, motivations and processes will be quite different.

This observation is similar to that made by Seddon *et al.* (1999), who introduce the 'IS Effectiveness Matrix'. They suggest that two key dimensions on which to base success measures are 'the type of system studied' and 'the stakeholder in whose interests the system is being evaluated'. FLOSS development typically occurs outside the corporate environments in which stakeholders are explicit and interests are often driven by the financial implications of the IS development. However, this does not mean that there are not stakeholders, such as core developers, peripheral developers, corporate users, individual users, and other whole projects who depend on the project whose success one is trying to measure. The conceptual model of success presented in this article intentionally provides a much greater emphasis on the inputs and the process of IS system development, and thus the developers, than traditional IS models of success. Studies of success in the FLOSS context should closely consider both the phenomena they are interested in (and thus relevant projects) and from which perspective they need to measure success, given their research interests. The development of a taxonomy of research interests and the identifications of the portions of the FLOSS universe appropriate for their study would be a useful task for future research.

5. CONCLUSIONS

This article makes a contribution to the developing body of empirical research on FLOSS by identifying and operationalizing success measures that might be applied to FLOSS. We developed and presented a theoretically informed range of measures which we think are appropriate for measuring the success of FLOSS projects, and we hope that these will be useful to researchers in the developing body of empirical research on FLOSS development. We complemented this theory development through an empirical study that demonstrated methods and challenges in operationalizing success measures using data obtained from SourceForge and made available to the community. This study demonstrated the portfolio approach to success measurement by taking measures from throughout the FLOSS development process and by using longitudinal data. The study allowed us to identify and communicate the limitations of our theory development and to elaborate areas that require particular care for researchers in this area.

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 and careful consideration as to which measures are most appropriate for the researcher's current research question should provide a better assessment of the effectiveness of each project.

While FLOSS is important 'for its own sake', it is also a form of system development growing in importance. There is substantial interest in learning from FLOSS, but such learning can proceed only when there is a firm understanding of the phenomenon, and understanding when it is working well is a crucial first step. Our future work will include more detailed analysis of both effective and ineffective projects. We plan to employ a theoretical sampling strategy based on a portfolio of relevant success measures to choose a few FLOSS development teams to study in depth, using both quantitative and qualitative research methods. 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.

ACKNOWLEDGEMENTS

This research was partially supported by NSF Grants 03-41475 and 04-14468. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors thank Chengetai Masango, Kangning Wang, Yeliz Esyerel and Qing Li for their contributions to the article.

REFERENCES

Arent J, Nørbjerg J. 2000. Software process improvement as organizational knowledge creation: A multiple case analysis. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS-33)*, Wailea, Maui HI. http://csd12.computer.org/comp/proceedings/nicss/2000/0493/04/04934045.pdf. Accessed 16 March 2006.

Basili VR, Caldiera G, Rombach HD. 1994. Goal question metric paradigm. In *Encyclopedia of Software Engineering*, Vol. 1. Marciniak JJ (ed.). John Wiley: New York, 528–532.

Boehm BW, Brown JR, Lipow M. 1976. Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering*, San Francisco, CA, 13–15 October, 592–605.

Butler B, Sproull L, Kiesler S, Kraut R. 2002. Community effort in online groups: Who does the work and why? In *Leadership at a Distance*, Weisband S, Atwater L (eds). Lawrence Erlbaum: Mahwah, NJ.

Chawla S, Arunasalam B, Davis J. 2003. Mining Open Source Software (OSS) Data using association rules network. In *Proceedings of the 7th Pacific Asia Conference on Knowledge Discovery and Data Mining (PACDD)*, Whang

K-Y, Jeon J, Shim K, Srivatava J (eds). Seoul, Korea, 461–466.

Conklin M. 2004. Do the rich get richer?: The impact of power laws on open source development projects. *Paper Presented at the Proceedings of Open Source 2004 (OSCON)*, Portland, OR.

Crowston K, Scozzi B. 2002. Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings Software* **149**(1): 3–17.

Crowston K, Annabi H, Howison J, Masango C. 2004. Effective work practices for software engineering: Free/Libre open source software development. In *Paper Presented at the WISER Workshop on Interdisciplinary Software Engineering Research, SIGSOFT 2004/FSE-12 Conference*, Newport Beach, CA.

Davis AM. 1990. *Software Requirements Analysis and Specification*. Prentice Hall: Englewood Cliffs, NJ.

Davis FD. 1989. Perceived usefulness, perceived ease of use and user acceptance of information technology. *MIS Quarterly* **13**: 319–340.

DeLone WH, McLean ER. 1992. Information systems success: The quest for the dependent variable. *Information Systems Research* **3**(1): 60–95.

DeLone WH, McLean Ephraim R. 2002. Information systems success revisited. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35)*, Waikola, Hawaii. http://csd12.computed.org/comp/proceedings/hicss/2002/1435/08/14350238.pdf. Accessed 16 March 2006.

DeLone WH, McLean ER. 2003. The DeLone and McLean model of information systems success: a ten-year update. *Journal of Management Information Systems* **19**(4): 9–30.

Diaz M, Sligo J. 1997. How software process improvement helped Motorola. *IEEE Software* **14**(5): 75–81.

Ewusi-Mensah K. 1997. Critical issues in abandoned information systems development projects. *Communication of the ACM* **40**(9): 74–80.

Ghosh RA. 2002. Free/Libre and open source software: survey and study. In *Report of the FLOSS Workshop on Advancing the Research Agenda on Free/Open Source Software*, Brussels, Belgium from http://www.infonomics.nl/FLOSS/report/workshopreport.htm. Accessed 16 March 2006.

Giuri P, Ploner M, Rullani F, Torrisi S. 2004. *Skills and Openness of OSS Projects: Implications for Performance*. Laboratory of Economics and Management, Sant'Anna School of Advanced Studies: Pisa, Italy, (Working paper).

González-Barahona JM, Robles G. 2003. Free software engineering: A field to explore. *Upgrade* **4**(4): 49–54.

Goranson HT. 1997. Design for agility using process complexity measures. *Agility and Global Competition* **1**(3): 1–9.

Gorton I, Liu A. 2002. Software component quality assessment in practice: Successes and practical impediments. In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, Orlando, FL, 555–558.

Grant RM. 1996. Toward a knowledge-based theory of the firm. *Strategic Management Journal* **17**: 109–122, (Winter).

Guinan PJ, Cooprider JG, Faraj S. 1998. Enabling software development team performance during requirements definition: A behavioral versus technical approach. *Information Systems Research* **9**(2): 101–125.

Hackman JR. 1987. The design of work teams. In *The Handbook of Organizational Behavior*, Lorsch JW (ed.). Prentice Hall: Englewood Cliffs, NJ, 315–342.

Hann I-H, Roberts J, Slaughter SA. 2004. Why developers participate in open source software projects: An empirical investigation. In *Proceedings of the Twenty-Fifth International Conference on Information Systems (ICIS 2004)*, Washington, DC, 821–830.

Hann I-H, Roberts J, Slaughter S, Fielding R. 2002. Economic incentives for participating in open source software projects. In *Proceedings of the Twenty-Third International Conference on Information Systems (ICIS 2002)*, Barcelona, Catalonia, Spain, 365–372.

Hertel G, Konradt U, Orlikowski B. 2004. Managing distance by interdependence: Goal setting, task interdependence, and team-based rewards in virtual teams. *European Journal of Work and Organizational Psychology* **13**(1): 1–28.

Howison J, Conklin M, Crowston K. In press. Flossmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*.

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

Kelty C. 2001. Free software/free science. *First Monday* **6**(12). http://www.firstmonday.org/issues/issue6-12/Kelty/index.html. Accessed 16 March 2006.

Krishnamurthy S. 2002. *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*. *First Monday* **7**(6). http://www.firstmonday.org/issues/issue7_6/krisnamurthy/index.html. Accessed 16 March 2006.

Lakhani KR, Wolf B. 2003. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. Retrieved 1 March, 2005, from http://opensource.mit.edu/papers/lakhaniwolf.pdf.

Lerner J, Tirole J. 2002. *Some Simple Economics of Open Source*. *The Journal of Industrial Economics* **2**(1): 197–234.

Mishra B, Prasad A, Raghunathan S. 2002. Quality and profits under open source versus closed source. In *Proceedings of the Twenty-Third International Conference on Information Systems (ICIS 2002)*, Barcelona, Catalonia, Spain, 349–363.

Mockus A, Fielding RT, Herbsleb JD. 2000. A case study of open source software development: The Apache server. In *Proceedings of the International Conference on Software Engineering (ICSE'2000)*, Limerick, Ireland, 263-274.

Rai A, Lang SS, Welker RB. 2002. Assessing the validity of IS success models: An empirical test and theoretical analysis. *Information Systems Research* **13**(1): 50–69.

Raymond ES. 1998. The cathedral and the bazaar. *First Monday* **3**: 3. http://www.firstmonday.org/issues3_3/raymond/index.html. Accessed 16 March 2006.

Reis CR, Fortes RP. 2002. An overview of the software engineering process and tools in the Mozilla project, In *Proceedings of the Open Source Software Development Workshop*, Newcastle Upon Tyne, UK, 155–175.

Scacchi W. 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings Software* **149**(1): 24–39.

Schach SR, Jin B Liguoy, Heller GZ, Offutt AJ. 2003. Determining the distribution of maintenance categories: Survey versus empirical study. *Empirical Software Engineering* **8**(4): 351–365.

Schach SR, Jin B, Wright DR, Heller GZ, Offutt AJ. 2002. Maintainability of the Linux Kernel, *IEE Proceedings-Software* **149**(1): 18–23.

Seddon PB. 1997. A respecification and extension of the DeLone and McLean model of IS success. *Information Systems Research* **8**(3): 240–253.

Seddon PB, Staples S, Patnayakuni R, Bowtell M. 1999. Dimensions of information systems success. *Communications of the Association for Information Systems* **2**(20): 61.

Shaikh M, Cornford T. 2003. Version management tools: CVS to BK in the Linux Kernel, *Paper presented at Taking Stock of the Bazaar: The 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*. Portland, OR. Available from http://opensource.mit.edu/papers/shaikhcornford.pdf. Accessed 16 March 2006.

Shenhar AJ, Dvir D, Levy O, Maltz AC. 2001. Project success: A multidimensional strategic concept. *Long Range Planning* **34**: 699–725.

Stamelos I, Angelis L, Oikonomou A, Bleris GL. 2002. Code quality analysis in open source software development. *Information Systems Journal* **12**(1): 43–60.

Stewart KJ, Ammeter T. 2002. An exploratory study of factors influencing the level of vitality and popularity of open source projects. In *Paper Presented at the Proceedings of the Twenty-Third International Conference on Information Systems (ICIS 2002)*, Barcelona, Catalonia, Spain, 853–857.

Stewart KJ, Gosain S. In press. The impact of ideology on effectiveness in open source software development teams, MIS Quarterly.