



University of
Massachusetts
Amherst

Brooks' Versus Linus' Law: An Empirical Test of Open Source Projects

Item Type	Article
Authors	Schweik, Charles M.;English, Robert
Download date	2026-04-22 14:08:09
Link to Item	https://hdl.handle.net/20.500.14394/36213



Brooks' Versus Linus' Law: An Empirical Test of Open Source Projects

Charles M. Schweik^{1,2,3}
Robert C. English^{2,3}

¹ *Department of Natural Resources Conservation*

² *Center for Public Policy and Administration*

³ *National Center for Digital Government*

University of Massachusetts Amherst

Amherst, MA USA

NCDG Working Paper No. 07-009

Submitted October 30, 2007

Support for this study was provided by a grant from the U.S. National Science Foundation (NSFIIS 0447623). However, the findings, recommendations, and opinions expressed are those of the authors and do not reflect the views of the funding agency. Thanks go to Megan Conklin, Kevin Crowston and the FLOSSmole project team (<http://ossmole.sourceforge.net/>) for making their Sourceforge data available, and for their assistance. We are also grateful to Thomas Folz-Donahue for programming work building our FLOSS project database, and for support from the National Center for Digital Government at the University of Massachusetts, Amherst (NSFIIS 0131923).

This paper will be presented at the Association for Public Policy Analysis and Management Conference, Nov 8-10th, 2007, Washington D.C.

BROOKS' VERSUS LINUS' LAW: AN EMPIRICAL TEST OF OPEN SOURCE PROJECTS

ABSTRACT

Free/Libre and Open Source Software (FOSS) projects are Internet-based collaborations consisting of volunteers and paid professionals who come together to create computer software. Because FOSS is available at no cost and has other advantages over proprietary software, both governments and businesses are increasingly using FOSS products. As a result, understanding what makes FOSS projects succeed has become a crucial question for many information technology (IT) professionals in both the public and private sectors. The specific question addressed in this paper is whether adding additional programming staff to FOSS projects makes those projects more successful. This question is controversial because traditional theory on software development indicated that adding programmers to late projects makes the problem worse (the famous Brooks Law in Software Engineering). We note that a similar theoretical argument regarding group size and public goods was made by Mancur Olson in his book *The Logic of Collective Action*. However FOSS literature suggests that adding programmers helps solve difficult programming problems (Linus' Law). We test these theories by studying the very large amount of data that is available about FOSS projects on Sourceforge.net (SF), an open source software hosting web site. In previous work, we created a measure of FOSS project success and then used this measure to classify nearly all the projects hosted on SF. In this paper, we examine whether the number of developers on a project correlated with success. The results showed that as the number of developers increases, the likelihood that a project will be successful increases. These results favor Linus' Law over Brooks Law/Olson. Although more research is needed, this finding suggests that there may be advantages in the way FOSS projects are organized and managed, both in software development but also in other areas of public sector work.

INTRODUCTION

Free/Libre and Open Source Software (FOSS) are Internet-based “commons,” where volunteers and paid professionals collaborate to produce software that is a public good. Over the last two decades, FOSS has had an active and growing community outside of the government sector, but in recent years, the FOSS phenomenon has been leading government IT managers to consider more fully open standards, interoperability of software and data, and the avoidance of vendor “lock in.” In addition, FOSS technologies provide some other potential benefits in government settings, including the potential to share software solutions between agencies and governments, the transparency of code (which in some domains, such as in voting systems, could be quite important), the potential reuse of software modules, a reduction in software acquisition costs, and the sharing of programming staff and resources across agencies or even governments toward some common goal (Wong, 2004; Ghosh et al., 2007).

Of course, FOSS is not the final answer in software, and we expect that now and into the future, digital government will be a mix of proprietary and FOSS solutions. But clearly both the use, and actual development of FOSS in public sector organizations is increasing, especially in countries outside the United States. For example, surveys conducted for the Dutch Government and for the European Union of public agencies reported significant FOSS use. Primary application domains include operating systems (GNU/Linux), Internet applications (Apache, PHP, mail servers), relational databases (MySQL), and desktop office suite software (Open Office) (Ghosh et al. 2007). In these surveys the researchers found a sizable number of “niche FOSS applications,” that have been implemented to solve specific tasks being undertaken by public organizations (Ibid., p. 12). Ghosh and colleagues conclude: “We therefore expect that public organizations that already are or will in the future become active [FOSS] producers will either focus their

activities around standard products that are already on the market and for which extensions, improvements, or new functionalities will be developed, or develop new software products for niche markets particular to the public sector.” Furthermore, this is probably true beyond the EU because other governments such as in Brazil, South Korea, and Thailand are emphasizing FOSS in their digital government policies (Wong, 2004).

In the United States, government IT groups appear to have shied away from generic policies which favor FOSS over proprietary software or vice versa, focusing more on the choice of “what solution appears to best do the job.” But there are signs that more attention is being given to FOSS as potential options, in part because of their adherence to open standards. One of the first cases of this was in Massachusetts, where a policy was being formulated to mandate open standards in the storage of office documents (MA ITD, 2004). Texas and Minnesota are moving in this direction (Gardner, 2007). In addition, consortia are emerging in efforts to increase FOSS-related collaborations across government IT shops, some more successful than others. Examples include a state and local-level initiative called the “Government Open Code Consortium” (GOCC.gov) and the Open Source Software Institute (<http://www.oss-institute.org/>), which is driven more by private sector firms with a focus on government client needs (Hamel, 2007). In the last three years there have been several annual conferences related to the use and development of FOSS software in U.S. Government settings (e.g, the Third Annual Government Open Source Conference, and the Third Annual Department of Defense Open Information Technologies Conference), and lastly, government FOSS project hosting sites like Free4Gov.org and GovernmentForge.net are emerging.

In short, Government agencies worldwide are both using more FOSS, and are showing interest in FOSS as a development paradigm for software products developed in their own shops or by contractors. However, one common concern over the use FOSS solutions is the longer-term

support of the projects. If there is no specific firm or organization that owns the software, many wonder how can it be assured that the software will continue to be maintained and supported. This is a concern voiced by IT managers in all sectors (see for example, Brunelli, 2006). The central question relates to the concern of “project abandonment” and what factors may ensure that it doesn't occur. From an operational standpoint, public IT managers interested in FOSS have only two options that might help avoid project abandonment. They can contribute some of their own programming staff to help the project move along, and to ensure in-house expertise, or, they can contribute financially to support programmers not on their staff who are working on or interested in working on the project. Either option tends to increase the number of developers on the project, thus helping to ensure its further development and maintenance.

This brings us to the key research question for this paper. *Are FOSS projects more likely to be successful if additional programming support is added to the project?* As we will describe in the next section, there are two competing and rather famous theories – Brooks' Law and Linus' Law – that predict opposite relationships between the number of developers on a team and project success. This paper empirically tests these competing theories.

We organize the paper as follows. First, we provide a short theoretical discussion related to the role of adding programming staff to the success of software projects. Second, we emphasize the importance and utility of the SF repository and associated data archiving efforts for answering these and other questions. Third, we build on previous work (English and Schweik, 2007) and present a success/abandonment measure for projects using the SF dataset. The result is a dataset of 107,747 FOSS projects that are classified as either successful, abandoned or indeterminate. Fourth, with this success/abandonment outcome variable established, we investigate the question: Does adding additional programmers to FOSS projects increase the probability of success? In this section we conclude that adding additional programmers increases the likelihood of project

success. We conclude with some reflections and next steps.

**COMPETING THEORIES:
“BROOKS' LAW” AND OLSON'S “LOGIC OF COLLECTIVE ACTION”
VERSUS “LINUS' LAW”**

The question about adding programmers to software projects to meet established goals has been studied for years. In traditional software engineering situations, where the software is developed by employees in firms, the concern over adding programmers is cost over-runs. These costs can be compounded when a proposed development schedule slips. Fred Brooks, in his oft-cited *The Mythical Man Month* ([Brooks, 1975] 1995: 25), argued that “adding manpower to a late project makes it later.” Factors that contribute to lateness include the communication complexities with new team members, coordination costs, and the training needed to bring them up to speed. However, Abdel-Hamid and Madnick (1990) have argued that this is not a universal truth; it is dependent on the other attributes of the project. For example, bringing on exceptionally skilled developers might make a difference.

What hasn't been recognized in much of the computer science literature is that a decade earlier, in another field of study, Mancur Olson (1965) made arguments similar to Brooks. Olson wasn't studying software development; his focus was on general “collective action” situations, where groups of people work to create public goods. A main thesis was that “the larger the group, the less likely it will further its common interests” (p. 36). Similar to Brooks, Olson argued that large groups face significantly higher costs in organizing (e.g., costs in communication, bargaining and in creating and maintaining formal organization) than in small group situations (p. 48).

It has been argued that Brooks' law doesn't translate to FOSS communities. In one of the most cited and influential early works on FOSS, *The Cathedral and the Bazaar*, Eric Raymond (2000a) presented “Linus' Law” (named after Linus Torvalds, lead developer of the Linux

operating system) by saying: “Given enough eyeballs, all bugs are shallow” (Raymond, 2000b). In other words, with a relatively large testing and developer community, a problem in software code will be identified quickly and a solution will be produced. And in the same paper, Raymond directly addressed the contradiction with Brooks' Law:

“The Brooks' Law analysis (and the resulting fear of large numbers in development groups) rests on a hidden assumption : ... that everybody talks to everybody else. But on open-source projects, the... developers work on what are in effect separable parallel subtasks and interact with each other very little; code changes and bug reports stream through the core group, and only within that small core group do we pay the full Brooksian overhead.”
(<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s05.html>).

The idea of the benefit of modularity in FOSS to reduce the costs of coordinating is supported by Jamie Zawinski of the Mozilla FOSS project, and Brian Behlendorf of the Apache and Collab.net projects. As Zawinski puts it: “Most of the larger open source projects are also fairly modular, meaning that they are really dozens of different, smaller projects. So when you claim that there are ten zillion people working on the Gnome project, your lumping a lot of people together who never need to talk to each other, and thus, aren't getting in each others' way” (Jones, 2000: 2). In short, modularity helps reduce coordination costs.

But in the same paper by Jones (2000), the question is raised as to whether Brooks' Law is even applicable to FOSS because of the lack of established schedule dates. Jones quotes Erik Troan of Red Hat (Linux), in saying “I don't know how you take a rule about 'making a late project later' and apply it to development which has very little scheduling.” But in our view, this isn't the main point of Brooks' Law. The point is that transaction costs make it harder for larger groups to coordinate. It isn't as much about slipping dates as it was a statement about the difficulties of coordination. As Olson's *Logic* (1965) argues, larger groups are harder to coordinate. Olson's theory also had no schedule dates attached to it.

Finally, Jones' article raises another issue, that isn't articulated this way, but goes against the applicability of Linus' Law. Citing an interview with Jamie Zawinski of the FOSS Mozilla Project, the point is made that in most FOSS projects, the majority of the code writing is done by a few dedicated programmers. The others contribute bug fixes or a “few hundred lines of code” here or there. In short, the idea of “more eyes” doesn't typically play out in FOSS. But this is a proposition that can be tested empirically. In general, if small core teams do most of the work, then projects with larger teams really shouldn't be more successful than projects with small teams.

In short, the discussion above leads us to three competing hypotheses about the relationship of developer team size and FOSS project success:

The Linus' Law Hypothesis: FOSS projects with larger development teams will be more successful (Linus' Law).

The Brooks' Law Hypothesis: FOSS projects with larger teams will face added coordination costs which will hinder FOSS development. Consequently, they will be less successful (Olson and Brooks).

The “Core Team” Hypothesis: The size of the development team won't have any effect on FOSS project success, because the core developer groups are almost always small teams (Zawinski).

The rest of this paper conducts an empirical examination of Sourceforge.net data to see which hypothesis is supported.

METHODS

The Importance and Utility of Sourceforge.net

There are a number of websites that act as FOSS project hosting sites and repositories, including BerliOS.org (<http://developer.berlios.de/>), Savannah.org (<http://savannah.gnu.org/>) and others (see http://en.wikipedia.org/wiki/Comparison_of_free_software_hosting_facilities).

Sourceforge.net (SF) is probably the largest of these sites, and as of October 2007, it hosts over 160,000 FOSS projects. While functions of hosting sites may vary, generally they provide an

online platform for project coordination and collaboration. SF provides for each project: a source code repository with version control functions, project communication forums, an error (bug) tracking facility, and project statistics and other general project information.

Because of the open nature of FOSS projects and of hosting sites like SF, anyone with a web browser can look up and find information about that project. Naturally, these project data are changing all the time. Developers with write or “commit” authority could be posting new changes to the version under production, or users may be downloading the software, posting bug reports, or contributing to project documentation. These projects are dynamic, and a visit to SF will capture a snapshot of the project status at that moment in time.

Two projects – one at the University of Notre Dame (<http://www.nd.edu/~oss/>) and the other based at Syracuse University (<http://ossmole.sourceforge.net/>) -- have realized the importance of capturing and keeping historical SF data. For researchers like us who are trying to understand the FOSS development phenomenon, the availability of these historical repositories are critically important. In a way, they are providing a kind of “remote sensor” of FOSS project activity. By collecting temporal “slices” of the SF database, these projects are building a historical repository of FOSS projects in a fashion similar to the way the Landsat satellite system and the EROS Space Center have collected and archived historical data about conditions and change on the earth's terrestrial surface. Given the likelihood that SF will be around for years to come (and hopefully these archival projects will be too), we think it is important to investigate what can be learned about FOSS using SF data alone.

To address the question of whether adding programmers helps or hurts FOSS projects, we need two things: (1) the number of developers associated with each project, and (2) a measure of FOSS project success and failure. The first is easy. SF has, for each project, a “number of developers” field. The second, a measure of project success or failure, is more difficult, but in the

next few sections, we present what we think are defensible methods to operationalize this variable using SF data.

Conceptualizing and Operationalizing Success and Failure (Abandonment) Using Sourceforge.net Data

There are at least two ways to conceptualize success and failure in FOSS projects: (1) direct use (Weiss, 2005; Crowston, Howison, and Annabi, 2006) or (2) developer activity. Successful projects would be ones where the software has a user base, and/or is exhibiting active development. Failure would be the opposite (no sign of use of the software or no developer activity). Both of these measures of success and failure have been used in previous work (e.g., Stewart and Ammeter, 2002; Capiluppi, Lago, and Morisio, 2003; Crowston, Annabi, and Howison, 2003; Hissam et al., 2007; Robles et al., 2005), and both are known to have limitations (Weiss, 2005). For example, regarding a use measure, is a project with very few *satisfied* end users, such as what might be found in highly technical, specialized bioinformatics analysis software, a failed project? We think not. Or should very specialized software performing highly specific and narrow functions and actively used (such as a file format conversion library) be classified as “failed” just because the developer team no longer works on a new release? We don't think this would be acceptable either. These complexities support Weiss' (2005: 7) contention that “the problem of defining and measuring success in open source is definitely not trivial.”

But if we wish to use SF data as a “remote sensor” of FOSS projects, some defensible definition and operationalization of project success and failure is needed that (1) accounts for the special circumstances above, and (2) can be readily built with SF data alone. In an earlier paper (English and Schweik, 2007) we developed such a classification, that we summarize here.

First, we argue that FOSS projects go through two main longitudinal stages: “Initiation”

and “Growth.” We define the *Initiation Stage* as the period where a project is active but there has yet to be a first public release of the code. We define the *Growth Stage* as the period after the first public release of code. Definitions of project success and failure differ between these two stages, and at any point in time, projects on SF will be in one of these two stages. We therefore need four measures: success and failure in Initiation, and success and failure in Growth. But instead of using the term “failed,” we prefer to use the term “abandoned.” In software, particularly in FOSS software, there are situations where the code is dropped, perhaps in incomplete form, but then perhaps used for some other project. From this perspective, could be argued that there are no failed projects, but rather just ones where the code is no longer utilized or maintained. Abandonment captures this concept better. With that point made, we have developed six classifications: *Success in Initiation* (SI); *Abandoned in Initiation* (AI); *Success in Growth* (SG); *Abandoned in Growth* (AG); *Indeterminate in Initiation* (II); and *Indeterminate in Growth* (IG). Table 1 presents both the definition and operationalization using SF data for each class.

Table 1: Six FOSS Success/Abandonment Classes and Their Methods of Operationalization (Adapted from English and Schweik, 2007)

<i>Class/Abbreviation</i>	<i>Definition(D)/Operationalization(O)</i>
Success, Initiation (SI)	D: Developers have produced a first release. O: At least 1 release (Note: all projects in the growth stage are SI)
Abandoned, Initiation (AI)	D: Developers have not produced a first release and the project is abandoned O: 0 releases AND ≥ 1 year since SF project registration
Success, Growth (SG)	D: Project has achieved three meaningful releases of the software and the software is deemed useful for at least a few users. O: 3 releases AND ≥ 6 months between releases AND does not meet the download criteria for tragedy detailed in the AG description below.
Tragedy, Growth (AG)	D: Project appears to be abandoned before producing 3 releases of a useful product or has produced three or more releases in less than 6 months and is abandoned. O: 1 or 2 releases and ≥ 1 year since the last release at the time of data collection OR < 11 downloads during a time period greater than 6 months starting from the date of the first release and ending at the data collection date OR 3 or more releases in less than 6 months and ≥ 1 year since the last release.
Indeterminate Initiation (II)	D: Project has yet to reveal a first public release but shows significant developer activity O: 0 releases and < 1 year since project registration

<i>Class/ Abbreviation</i>	<i>Definition(D)/Operationalization(O)</i>
Indeterminate Growth (IG)	D: Project has not yet produced three releases but shows development activity or has produced 3 releases or more in less than 6 months and it has been less than 1 year since the last release. O: 1 or 2 releases and < 1 year since the last release OR 3 releases and < 6 months between releases and < 1 year since the last release

Admittedly, these definitions and operationalizations are not perfect. For example, our choice of “one year without a release” as a proxy for abandonment could introduce classification error because some projects without a release in the last year might be actively being developed. Or our conservative operationalization of a “useful software product” as a project with “more than ten downloads in at least a six month period starting at the date of the first release” could result in classifying projects as useful, when they are not, because the software may have been downloaded ten times by people were curious about the software but not actively using it. We will return to these issues in the classification validation section below.

A Success/Abandonment Classification of SourceForge.net Projects

We utilized data from the SF repository collected by the FOSSMole project in August, 2006. FOSSMole's collection had most of the data we needed, however it was missing the number of releases and dates of first and last release which were required to operationalize our concepts above. Consequently, we “crawled” SF ourselves in the fall of 2006 to fill in these data gaps. Our final result was a 107,747 SF project dataset representing SF in August-October 2006 with all the parameters we needed for the classification in Table 1. Note that we do not classify Successful projects in the Initiation Stage (SI), because this class consists of all the projects in the Growth Stage. The results of this classification are shown in Table 2.

<i>Table 2: SF Success/Abandonment Classification Results (August-October 2006 Data)</i>	
<i>Class</i>	<i># of Projects (%of Total)</i>

Abandoned in Initiation Stage (AI)	37,320 (35)
Success in Growth Stage (SG)	15,782 (15)
Abandoned in Growth Stage (AG)	30,592 (28)
Indeterminant in Initiation Stage (II)	13,342 (12)
Indeterminant in Growth Stage (II)	10,711 (10)
Total	107,747
Note: Projects that are Successful in the Initiation Stage (SI) are not listed because these are Growth Stage projects. Including SI would double count.	

Testing the Validity of the Classification

There could be false classifications in the Table 1 results, in part because of the issues mentioned earlier, so before proceeding further, we wanted to go through a validation process to assess the accuracy of our classification results. We took a random sample of three hundred classified projects, and three people over the course of a month checked projects' classification results by manually reviewing SF pages. The validation results are presented in Table 3.

**Table 3:
FOSS Project Classification Validation Results**

Original Classification Class	Correct	Incorrect	Deleted Project or Missing Data*	Totals	Error Rate in % (95% confidence interval in parenthesis)
Abandoned in Initiation (AI)	77	10	19	106	11.5 (4.7-18.3)
Abandoned in Growth (AG)	93	8	0	101	7.9 (2.5-13.3)
Successful in Growth (SG)	92	0	1	93	0
Totals	262	18	20	300	6.4 (3.5-9.3)

* 18 projects were deleted from SF and only 2 had missing data; (2) All projects that were deleted from SF were classified as AG; (3) If the projects deleted from SF are considered to be abandoned then the Error Rate for Class AI is 9.5%

Explaining Abandoned in Initiation Classification Error. In this class, nineteen projects which existed in the fall of 2006 could not be validated. Of these nineteen projects, one was missing key information needed to make a manual classification. The other eighteen projects had been deleted from SF between the time our original data was collected (August 2006) and the time we gathered our validation data (Spring 2007). All of these eighteen deleted projects were classified as AI in our original classification. SF regularly purges inactive projects – the likely cause of this problem. Consequently, it is highly likely that most, if not all, of these projects were classified correctly. If we assume that all the deleted projects were in fact abandoned, then our error rate for the AI class would be reduced from 11.5 to 9.5 percent with a 95 percent confidence interval ranging from 3.8 – 15.2 percent. But there is a rival explanation -- that some or all of these eighteen projects were successful and were moved to other hosting platforms by the project developers. So in Table 3, we leave the error rate at 11.5 percent.

What could have caused the other ten projects assigned AI (row 2, column 3 in Table 3) that had full information to be classified incorrectly? Recall that our definition of AI was “abandoned before producing a public release.” No activity and no first public release is relatively easy to diagnose, but what constitutes a “reasonable amount of time”? We set it to one year based on sampling SF data. The ten projects misclassified were projects that did not list a release for a year after they were registered on SF, but showed some activity in the year before our data were collected or had produced a first release either before or after our data were collected. Projects hosted on SF occasionally release their files on a website separate from SF without listing the releases on the SF website.

Explaining Abandoned Growth Classification Error. The error rate in Table 3 for the AG class appears to be approximately equal to the error rate for the AI class. As in the AI class, all the projects incorrectly classified in the original classification turned out to be not abandoned.

This agreement between error rates in the AG and AI class may indicate that a relatively consistent proportion of projects still have activity after not producing a release over a time period of one year.

Explaining Successful Growth Classification Error. As Table 3 shows, we found no misclassified projects in the SG class, which may be the class of most interest to some readers. And we should remind readers that there is no Successful Initiation class listed, because all projects in the Growth Stage are these successful cases.

The results of this validation show that the classification varies from what we would consider “reasonably accurate” (AI incorrect between 11.5 and 9.5% of the time) to “extremely accurate” (Successful Growth being 100% accurate). This validation exercise provides a high-level of confidence that future analysis based on this classification will produce meaningful results.

WHICH IS SUPPORTED? THE LINUS' LAW, BROOKS' LAW OR “CORE TEAM” HYPOTHESIS?

Three hypotheses were presented earlier: (1) The Linus' Law Hypothesis, that predicts that larger FOSS teams with “many eyes” will have a high probability of success; (2) The Brook's Law Hypothesis (supported by Olson's (1965) *The Logic of Collective Action*), which suggests that larger FOSS projects will face significant coordination costs and consequently, a lower probability of success compared to smaller team projects; and (3) The “Core Team” Hypothesis, which argues that FOSS projects have small sets of “core” programmers that do most of the work, regardless of the team size. The Core Team Hypothesis suggests that the size of the developer team should make little difference in determining success or abandonment. In other words, all projects generally are small teams.

Recall that in the FLOSSMole dataset, there is the field “dev_count” which provides the

number of developers associated with each FOSS project stored in the SF repository. With our success/abandonment classification established, we can investigate the relationship between development team size and FOSS project success.¹

The analysis we present next leads us to three main findings: (1) Projects that list large development teams tend to fall in the Successful Growth class; (2) A relatively small percentage of the total number of Successful Growth cases hosted on SF list over ten developers; and (3) the empirical evidence supports the Linus' Law hypothesis. Adding programmers increases the likelihood of a project being successful. Below we describe how we came to each of these conclusions.

Larger development team projects tend to be Successful Growth cases

Figure 1 graphs the number of projects (log scale), and the associated number of developers (log scale) for each determinate class (Abandoned in Initiation-AI, Abandoned in Growth-AG, and Successful in Growth-SG). As seen in Figure 1, projects with more than 40 developers tend to be in the Successful Growth class. In other words, abandoned projects in either Initiation or Growth stages tend to have a smaller number of developers associated with them when they become abandoned.

*** Figure 1 about here ***

A relatively small percentage of Successful Growth stage projects are large teams

¹ To some readers, looking at number of developers on a project versus its success or abandonment might be puzzling. Isn't the definition of an abandoned project one that has no developers? The answer, in this context, is no. Recall that our abandonment measures described earlier for the Initiation and Growth stages are based upon project *release activity over a period of time*, and do not use the developer count field.

While Figure 1 shows that the SG class has more projects that list large developer teams than the abandoned classes, Figure 2 shows that only a small percentage of SG projects are large teams. In September 2006 when we collected this SF data, there were 15,782 projects in total that classify as SG. Figure 2 shows that over 80% of these SG projects had five developers or less, and just over 90% had ten developers or less. Roughly 10% or about 1,500 SG projects have teams of 11 or more developers. A relatively small number in terms of all FOSS projects in SF, but still a sizable number.

*** Figure 2 about here ***

What can be concluded from this? It raises the question of how representative SF is of the FOSS community as a whole. It could be that some projects with larger development communities move onto their own project management sites. But we suspect that SF is probably fairly representative of the broader FOSS project community. The analysis above suggests that the majority of FOSS developers *working on successful projects* are working in relatively small teams. This likely has implications about FOSS project governance and institutional design. Smaller teams are likely to have less formal development rules. And it may mean that most successful growth projects have less formal institutional structures that operate more on informal mechanisms and social norms. But it also suggests that a small percentage of them probably require some more complex system of governance and management.

Which hypothesis is supported? Linus, Brooks/Olson or Core Team?

To explore this question, we utilized simple logistic regression using Developer Count as a single independent variable against our dichotomous dependent variable “Abandoned in the

Growth Stage (0) and Successful in the Growth Stage (1)”. In our dataset, 46,374 projects fell in these two categories. The results are presented in Table 4.

Table 3: Estimated Coefficients, Standard Errors, z-Scores, Two-tailed p-values, and 95% Confidence Intervals for Simple Logistic Regression of Number of Developers Influence on Project Success or Abandonment (n: 46,374)						
Variable	Coeff.	Std. Err.	z	P> z 	95% CI	
Developer count	0.21084	0.00473	44.58	<2e-16***	.202	.220
Constant	-1.14993	0.01440	-79.88	<2e-16***	-1.18	-1.12
Significance codes: *** 0.001 C: 0.638; R ² : 0.088						

The results in Table 3 show that the number of developers on a project is a statistically significant predictor of FOSS success. The odds ratio for the Developer Count coefficient is 1.24. This indicates that for each developer added to a project, the odds that the project will become successful in the growth stage increases 1.24 times.

To help make this interpretation clearer, Table 4 presents the probability of a project being Successful Growth versus Abandoned Growth for a representative range of SG and AG projects as predicted by the coefficients in Table 3. For comparison to the actual data that produced these predicted probabilities, we calculated the actual percentage of SG projects in our dataset. Since showing all developer counts would produce a very cumbersome table, Table 4 shows an excerpt of data for developer counts incremented by 5. All projects with greater than 86 developers have a predicted and an actual probability of 1.

Table 4. Predicted Probabilities for Successful Growth Based on Developer Counts		
Developer Count	Predicted Probability of SG	Actual Probability SG (SG/SG+AG) in our SF dataset
1	0.2810838	.255
6	0.528742	.582
11	0.763016	.737
16	0.902338	.790
21	0.963655	.893
26	0.987027	.923
31	0.995441	.875
36	0.998407	1.0
41	0.999444	1.0
46	0.999806	1.0
51	0.999933	1.0
56	0.999977	1.0
61	0.999992	1.0
66	0.999997	1.0
71	0.999999	1.0
76	1.0	1.0
81	1.0	no project with 81 developers
86-340	1.0	1.0

We fully recognize that the simple univariate regression model provided in Table 3 does not fully explain the factors that lead FOSS projects to success or abandonment. Obviously, a multivariate model that includes other theoretically-driven covariates capturing factors like project financing, for example, is needed to improve the model's goodness of fit. But the point for this paper is that even this simple model does a statistically impressive job of choosing one hypothesis over another. Linus' Law is supported, and Brooks' Law and the Core Team hypothesis are rejected in the context of FOSS projects in SF. Our conclusion: adding more programmers improves the chances that a FOSS project will be successful.

CONCLUSION

Our finding creates a kind of chicken-egg problem. Do more developers lead to project success? Or is it that successful projects attract more developers, in part because of the economic motivations that drive some programmers to participate (e.g., signalling programming skill to a broad community, self-learning through reading other's programs and peer-review) (Lerner and Tirole, 2005)? Our hunch is that both occur, but the fact that the rise in success rate holds strongly for projects with smaller numbers of developers tends to favor the idea that more developers produces success. We can't answer this question with the dataset we have, because it represents generally one point in time (August-October, 2006). It may, however, be possible to answer this using the other "FOSS remote sensor" -- the Notre Dame (ND) longitudinal dataset. By sampling the ND data at different time periods, running our classification procedure for projects at these different points in time, and then comparing the number of developers versus project class over time, it may be possible to separate out these dynamics.

But even with the above limitations acknowledged, the findings above lend strong support to the arguments made by Raymond: that the relatively flat, modular system of coordination in FOSS projects allows the addition of programmers without too much impact regarding coordination costs. So the real concern in FOSS appears to be not the issue of slowing projects down with the addition of more programmers, but rather, given the large number of small developer team projects, getting "more eyeballs" participating in the projects.

Implications for public sector organizations? We see at least two.

First, from a FOSS user perspective, it may be worth checking the project website to see how many developers are associated with the project before choosing a particular product. If there

is a relatively large number of developers, that's a good sign. Of course, that shouldn't be the sole criteria—there will still be lots of successful small projects. Nevertheless, development team size should be one of many selection criteria.

Second, we noted in the introduction that it appears more government agencies are either directing their own staff to program new functions to existing FOSS projects based on their needs (e.g., adding a new function to a content management system) or considering the FOSS development paradigm for situations where they want to develop their own software from scratch. The research here suggests that adding programmers to help existing FOSS projects is beneficial to these projects. It also suggests that in situations where a new project is being initiated using public sector programming or contractual staff, it might be worthwhile to actively look for other development partners within the agency or even externally (other agencies or governments) who might have programming staff to contribute to the project, as long as some consideration is made related to the architecture (e.g., modularity) of the project.

Finally, the findings here may have broader implications outside of software in how public organizations may wish to organize work in the future. Recently, we are seeing the emergence of “open source like” collaborations in other public sector work, sometimes in surprising places (for an example in intelligence gathering for national security, see Andrus, 2005). While more research is needed on to how FOSS projects are organized and managed, these findings suggest that flatter hierarchies and modularity create some advantages in the way some work can be accomplished.

REFERENCES

- Abdel-Hamid, T.K., and Madnick, S.E. 1990. “The Elusive Silver Lining: How We Fail to Learn from Software Development Failures.” *Sloan Management Review* 32, 1: 39-48.
- Andrus, D. C. 2005. "The Wiki and the Blog: Toward a Complex Adaptive Intelligence Community." *Studies in Intelligence*, Vol 49, No 3.

- Brooks, F. P. Jr. [1975] 1995. *The Mythical Man-Month: Essays on Software Engineering*. Anniversary Edition. Reading, MA: Addison-Wesley.
- Brunelli, M. 2006. Oracle users concerns with open source support. SearchOracle.com. August 15, 2007.
- Capiluppi, A., Lago, P. and Morisio, M. 2003. "Evidences in the Evolution of OS projects through Changelog Analyses," In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering. http://www.publicsectoross.info/images/resources/15_154_file.pdf <http://opensource.ucc.ie/icse2003>. Accessed Dec 12, 2006.
- Crowston, K., Annabi, H. and Howison, J. 2003. "Defining Open Source Project Success," In Proceedings of the 24th Int.l Conference on Information Systems, ICIS, Seattle, 2003.
- English, R. and Schweik, C.M. 2007. "Identifying Success and Tragedy of Free/Libre and Open Source (FLOSS) Commons: A Preliminary Classification of Sourceforge.net projects." Proceedings of the 29th International Conference on Software Engineering + Workshops (ICSE-ICSE Workshops 2007), 20-26 May, Minneapolis, Minnesota.
- Gardner, W.D. 2007. OpenDocument Software Gains Momentum. Channel Web Network. <http://www.crn.com/software/197004139>. Accessed October 18, 2007.
- Ghosh, R.A., Glott, R., Gerloff, K., Schmitz, P-E., Aisola, K., and Boujraf, A. 2007. "Study on the Effect on the Development of the Information Society of European Public Bodies Making Their Own Software Available as Open Source: Final Report." http://www.publicsectoross.info/images/resources/15_154_file.pdf. Accessed October 18, 2007.
- Hamel, M.P. 2007. Open-Source Collaboration in the Public Sector: The Need for Leadership and Value. NCDG Working Paper #07-004. http://www.umass.edu/digitalcenter/research/working_papers/07_004HamelOSCollaboration.pdf. Accessed October 18, 2007.
- Hissam, S., Weinstock, C.B. , Plaksoh, D. and Asundi, J. 2007. Perspectives on Open Source Software. Technical report CMU/SEI-2001-TR-019, Carnegie Mellon University. 10 Jan. 2007, <http://www.sei.cmu.edu/publications/documents/01.reports/01tr019.html>.
- Koontz, Tomas M., Toddi A. Steelman, JoAnn Carmin, Katrina Smith Korfmacher, Cassandra Moseley, and Craig W. Thomas. 2004. *Collaborative Environmental Management: What Roles for Government?* Washington, D.C: Resources for the Future Press.
- Krishnamurthy, S. 2002. "Cave or Community: An Empirical Examination of 100 Mature Open Source Projects" *First Monday*. Vol 7, No. 6. http://www.firstmonday.org/issues/issue7_6/krishnamurthy/
- Lerner, J., and Tirole, J. 2005. Economic Perspectives on Open Source. In Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. R. (eds.) *Perspectives on Free and Open Source Software*.

Cambridge, MA: MIT Press. pp. 47-78.

MA ITD, 2004. Commonwealth of Massachusetts Information Technology Division, Open Standards Policy. Available at http://www.mass.gov/Aitd/docs/policies_standards/openstandards.pdf.

Olson, M. 1965. *The Logic of Collective Action: Public Goods and the Theory of Groups*. Cambridge, MA: Harvard University Press.

O'Mahony, S. 2005. Nonprofit Foundations and Software Collaboration. In Feller, J., Fitzgerald, B., Hissam, S.A., and Lakhani, K.R. (eds.) *Perspectives on Free and Open Source Software*. Cambridge, MA: MIT Press.

Raymond, E. 2000a. "The Cathedral and the Bazaar." <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s05.html>. Version 3.0. Accessed 10/20/2007.

Raymond, E. 2000b. "Release Early and Release Often." In "The Cathedral and the Bazaar." <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html>. Version 3.0. Accessed 10/20/2007.

Robles, M., Gonzalez, G., Barahona, J.M., Centeno-Gonzalez, J., Matellan-Olivera, V. and Rodero-Merino, L. 2005. "Studying the Evolution of Libre Software Projects Using Publically Available Data," In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*, 12 Dec. 2006. <http://opensource.ucc.ie/icse2003>.

Schweik, C.M. and English, R. 2007. "Tragedy of the FOSS Commons? Investigating the Institutional Designs of Free/Libre and Open Source Software Projects". *FirstMonday*. Vol 12. No. 2. http://www.firstmonday.org/issues/issue12_2/schweik/index.html.

Stewart, K.J. and Ammeter, T. 2002. "An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects," In L. Applegate, R. Galliers, and J.I. DeGross (eds.) *Proceedings of the 23rd International Conference on Information Systems*, Barcelona, 2002, pp. 853-57.

Weiss, D. 2005. "Measuring Success of Open Source Projects Using Web Search Engines," *Proceedings of the First International Conference on Open Source Systems*, Genova, 11th-15th July 2005. Marco Scotto and Giancarlo Succi (Eds.), Genoa, 2005, pp. 93-99.

Wong, K. 2004. *Free/Open Source Software: Government Policy*. United Nations Development Programme-Asia Pacific Development Information Programme (UNDP-APDIP).

Figure 1.

Number of Developers versus Number of Projects for each Class



