Application of Survival Model to Understand Open Source Software Release

Ravi Sen
Department of Information and Operations Management
Texas A&M University
rsen@mays.tamu.edu

Matthew L. Nelson
Department of Accounting and Business Information Systems
Illinois State University
mlnelso@ilstu.edu

Chandrasekar Subramaniam
Department of Business Information Systems and Operations Management
University of North Carolina at Charlotte
csubrama@uncc.edu

Abstract

One of the recurrent themes in open source software research is to understand the impacts of various project characteristics on its success. Open source software (OSS) projects rely on voluntary participation of developers and tend to be continually in development. Hence, an important measure of success is the time it takes for an OSS project to release a stable version to its users. However, there is little research on this success measure and how the OSS characteristics enable or delay the progress towards stable release. In this study, we use survival analysis technique on open source project data to explore the impacts of OSS characteristics on the time it takes to release stable software versions. We find that when compared to the interest of developers in the project, interest of end-users has a greater positive effect on an OSS project progress towards stable release. Our findings also suggest that the use of C and C-like programming languages or a Weak-Copyleft license for the open source project negatively impact the project's time to reach stable status. In OSS projects less than 8 months since becoming public, the use of a Strong-Copyleft license positively affects the project's progress. One of the implications of our findings is that OSS project administrators should control software change requests or form smaller developer groups to better control the delays due to higher developer interest in their projects.

Keywords: Open source project, OSS, FLOSS, OSS popularity, OSS success, survival, hazard, ordinal regression

Introduction

The widespread adoption of open source software (e.g. Apache, Sendmail, various flavors of Linux) has generated immense interest among academics who want to understand and explain various aspects of this phenomenon (Nelson, Sen, & Subramaniam, 2006). The open and voluntary approach to development in open source software (OSS) is arguably more efficient than the development methods of proprietary software (Martin, 1998) and implements a voluntary form of concurrent design and testing of software modules (Kogut & Metiu, 2001). One of the recurrent themes in open source software research is to understand the impacts of various project characteristics on project success and researchers have offered different measures of open source project success. Knowledge about the OSS success measures and success predictors can help to evaluate legal and policy decisions on this competing model of software development. In addition, this knowledge plays an important role in helping administrators better manage release timing and attract talented developers, sponsors, and end-users to their OSS projects (Hann, Robert, & Slaughter, 2004).

The OSS literature has identified several success measures such as project activity levels, development team size, and time taken to fix software bugs (Crowston, Howison, & Annabi, 2006). Given that OSS software tend to be continually in development, there is relatively very little understanding of a key measure of OSS success - the project's progress (Crowston, Annabi, & Howison, 2003). Open source software (OSS) projects also are known to suffer from resource constraints, since most such projects have very few developers working on them. Many OSS projects try to survive and sustain development work by relying on voluntary donations from users. Several do not have the experts such as usability experts, documentation writers, etc., to help improve their final product. Despite these constraints, open source projects still have to compete with commercial software producers who have more resources at their disposal. In order to compete effectively, open source software projects need to develop and release stable versions of their product early and often. Since software benefit from network effects, it is important that the stable version is released as early as possible to leverage first-mover advantages inherent in network products. However, there are no studies, particularly empirical, that we know of which investigate OSS projects' progress towards stable release. In this paper, our objective is to understand the effects of *an OSS project's characteristics on the time taken to release a stable version of the OSS after it has been made public (i.e., after it has been registered in the leading open source software repository SourceForge (www.sourceforge.net)).* Our study uses survival analysis and the framework of OSS success derived from DeLone and McLean's IS success model (DeLone & McLean, 2003)

Among the several interesting findings, are two in which the levels of developer interest and user interest have a negative impact on the project's progress in the early days of the project (i.e., it takes longer to reach a stable status). As the time from the project registration increases, developer-interest and user-interest have positive impacts on the time to release stable versions. We also find that the choice of Weak-Copyleft license for the OSS project increases the time taken by the project to reach stable status, while a Strong-Copyleft license results in faster progress to a stable status during the first 229 days of the OSS project.

The rest of our paper is organized as follows. We review the related literature in the next section, followed by our research hypotheses in section 3. In section 4 we present the Extended Cox hazard model used for our study. We then describe the data, model estimates and the results of hypotheses testing. We conclude the paper

with a discussion of our results and managerial implications.

Review of Related Literature

Several noteworthy trends in the OSS arena have emerged and should be highlighted at the outset. The notion that OSS development efforts are best represented by a small fraction of technical experts has long passed, as users and developers of all types increasingly integrate open source software into IT solutions. OSS has experienced widespread mainstream adoption, with predictions reaching as high as 80% of all commercial software packages to include some elements of open-source technology in 2012 (Driver, 2010). As organizations strive to reach a balanced software portfolio, the breadth of OSS solutions are expanding from horizontal support solutions (operating systems, web browsers) towards vertical solutions (functional, business unit specific applications).

Although the longer-term implications of these OSS trends remain to be seen, some noteworthy consequences have become clear. For example, OSS development is shifting towards ensuring that a final "whole product" is completed and released, with greater emphasis placed on project progression success factors and final project outcomes (Fitzgerald, 2006; Driver, 2010; Bardhan, Kauffman, & Naranpanawe, 2010). Greater structure is needed in the OSS development process, leveraging fundamental project management techniques, and with greater focus towards the *measurement and timing* of defined outcomes. IT industry analysts are encouraging companies to launch formal enterprise wide open source software governance programs, to better manage complex licensing arrangements, integrate findings into baseline service level agreements (SLAs) and to better prepare for ramifications of mergers and acquisitions (Fitzgerald, 2006; Driver, 2010; Bardhan et al., 2010). Another notable consequence is a shift towards less emphasis being placed on developer *skills* and greater emphasis being placed on the *extent* of end-user involvement and the *degree* of developer participation in OSS projects (Fitzgerald, 2006; Driver, 2010).

Understanding the objective measures of OSS project success is important since it helps OSS project managers to evaluate their projects and take steps to meet the project goals (Crowston et al., 2006). The literature on OSS proposes measures of success of OSS projects from perspective of the OSS development process and which could complement the traditional success measures. One study has identified project activity level, development team/community size (i.e. number of active contributors to the project), and time taken to fix software bugs as key measures for OSS project success (Crowston et al., 2003). Another study has identified as success measures the extent to which a project attracts input from the development community (e.g. number of developers), and the extent to which it produces observable outputs such as the addition of new features to the software or the fixing of software bugs (Stewart, Ammeter & Maruping, 2006). Users' interest over time (i.e. change in the number of subscribers to an OSS project) and the amount of development activity (i.e. the number of files released) have also been used as measures of OSS project success (Stewart et al., 2006). Finally, given the large number of abandoned information systems projects (Ewusi-Mensah, 1997), the completion of a project may be an important measure of success. Howison and Crowston (2004) suggest that the progress of the OSS project to a stable status can be a proxy for project completion since most OSS projects are always in development.

The "project progress as a measure of success" argument is also supported by software engineering literature, which identifies software attributes such as completeness, consistency, testability, usability, and reliability as measures of

software quality (e.g. Bardhan et al., 2010; Gorton & Liu, 2002). Since these attributes improve as the software progresses towards a stable state, the ability to release a stable/mature version of the software under development is considered a useful indicator of project success (Crowston et al., 2003; Mockus, Fielding, & Herbsleb, 2002). OSS development projects, when compared with commercial software development, can present project resource challenges (financial, human and timeline), in addition to the reliability and accessibility of those resources. The resource constraints can result in delayed release of a stable version of the software. Hence, how quickly the software reaches a stable and usable state is a good success measure (Crowston & Scozzi, 2002).

To understand the predictors of OSS success, the DeLone and McLean's model of information systems (IS) success is the most commonly used in OSS research (Crowston et al., 2006). The DeLone and McLean's model suggests six interrelated factors for system success – system quality, information quality, use, user satisfaction, individual impact and organizational impact (DeLone & McLean, 2003). However, these conventional measures focus on the use and the *use environment* of the software. In the case of OSS, the use environment is difficult to observe while the development environment is more publicly visible (Crowston et al., 2006). Hence, other measures may be useful in OSS to complement traditional software success measures. The literature on OSS proposes measures of success of OSS projects from perspective of the OSS development process and which could complement the traditional success measures. One study has identified project activity level, development team/community size (i.e. number of active contributors to the project), and time taken to fix software bugs as key measures for OSS project success (Crowston et al., 2003). Another study has identified as success measures the extent to which a project attracts input from the development community (e.g. number of developers), and the extent to which it produces observable outputs such as the addition of new features to the software or the fixing of software bugs (Stewart et al., 2006). Users' interest over time (i.e. change in the number of subscribers to an OSS project) and the amount of development activity (i.e. the number of files released) have also been used as measures of OSS project success (Stewart et al., 2006).

The OSS literature has also identified several predictors of OSS success. These predictors are the characteristics of the OSS projects and the characteristics of the key stakeholders involved (i.e., developers and end-users). Bonaccorsi and Rossi (2003) suggest that server-based OSS projects, such as Apache web server, are more successful than client-based OSS projects, such as Linux. Other researchers find that the degree and nature of network embeddedness of an OSS project impact its success and that greater embeddedness does not always result in project success (Grewal, Lilien, & Mallapragada, 2006). Stewart et al., (2006) studies sponsored and non-sponsored OSS projects and found that non-restrictive licenses in general increase end-user interest in the projects than restrictive licenses. Lerner and Tirole (2005) show that OSS applications geared toward end-users and system administrators have restrictive licenses while those aimed at developers have less restrictive licenses. Subramaniam, Sen and Nelson (2009) show that the effects of restrictive licenses on project activity is somewhat nuanced. The adverse impact of license restrictiveness on project activity holds only if the target audience for the OSS project is other developers and not when they are system administrators. Other project factors related to OSS success are the operating system platform and the underlying programming language of the OSS project (Subramaniam et al., 2009). One of the defining characteristics of an OSS project is the voluntary participation of developers in creating, debugging and maintaining the

software resulting from the project. Hence, some of the OSS success factors identified in the literature relate to the developers themselves, such as developer motivation and interest (Bonaccorsi et al., 2003) and the presence of a critical mass of developers in the project (Mockus et al., 2002). While the above predictors have received attention in OSS studies, to the best of our knowledge, there are no studies which investigate these predictors' impacts on an OSS project's progress towards releasing a stable version of the software. Our study will help to fill this gap and add to our understanding of the dynamics of OSS project management and success.

Model and Hypotheses

In this section, we discuss the research model (Figure 1) and the hypotheses. As presented in the literature review, the DeLone and McLean (2003) success model was updated by OSS scholars to take into account the more publicly visible development environment of open source projects. User interests and developer interests were used to indirectly measure the information quality and system quality. In our paper, we borrow from these updated research by Crowston et al (2006), Stewart et al (2006), and Subramaniam et al (2009). We begin the OSS project progress which is the dependent variable in our model.

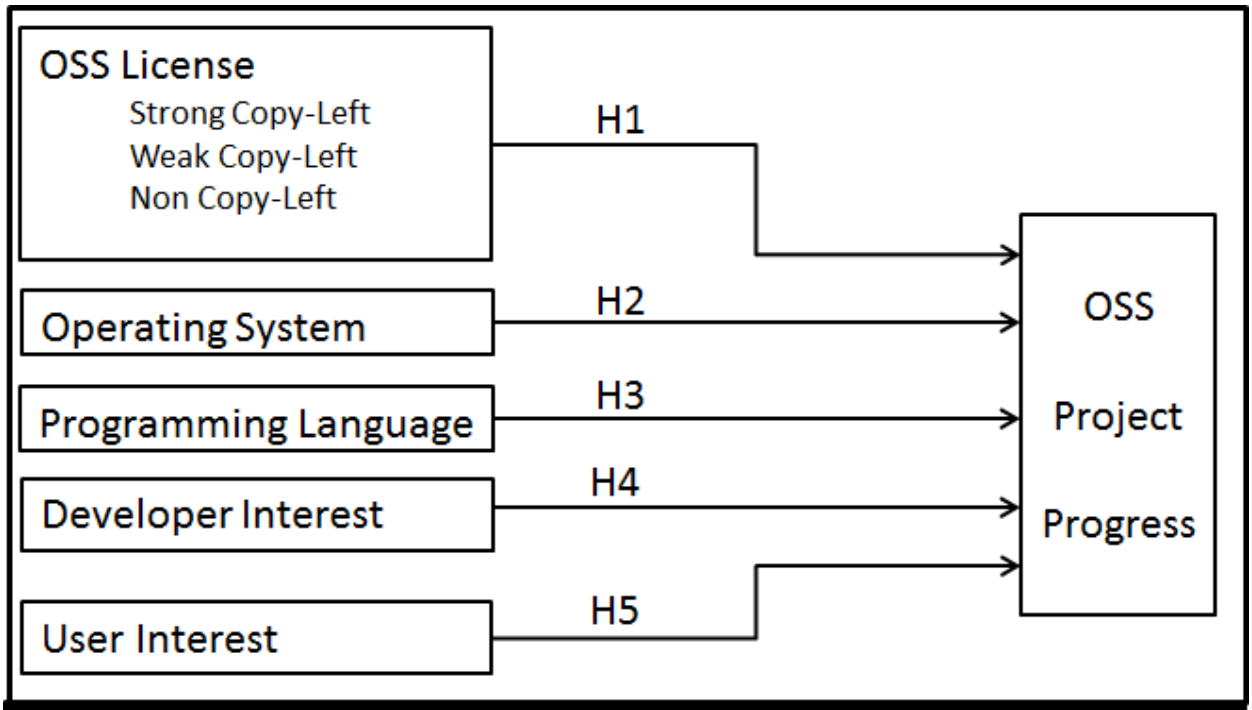

Figure 1: Theoretical Model

OSS Project Progress: There are several ways to identify an OSS project's development status in order to assess its progress. Based on system development life cycle principles, a software project can be in one of five stages - Requirements Planning, Analysis, Design, Development, and Maintenance (Hoffer, George, & Valacich, 2008). Projects in the later stages of the life cycle are considered closer to stable/mature status than projects in the earlier stages. Sourceforge.net, a repository of information about OSS projects and the source for the empirical data in our study, uses the following stages of project status: Planning, Pre-Alpha, Alpha, Beta,

Production/Stable, Mature, and Inactive. The development status of each project is reported by its project administrator(s) and Sourceforge allows selection of projects by their status. The projects in stages Production/Stable and Mature are considered stable projects and have the best chance to build a user community around them (Krishnamurthy, 2002). Though the stages are important measures of progress, the time to reach that stage is important as well. Unike commercial software development settings, OSS projects are sustained by efforts of volunteer programmers who contribute to projects concurrently (or at separate times) and often do so in their spare time (between paid projects or in down-cycles). Similar challenges are experienced with managing other OSS project resources such as with financial, technical and intellectual property. Collectively these constraints can increase the possibility of OSS project delays and further underscore the relevance and importance of understanding time to release a stable version of software. For most projects, Sourceforge also reports the date of project registration and the date of the most recent file release. In section 4.0 (on method and data collection), we provide additional explanation for use of the project status scale and dates provided by Sourceforge.

Predictors of Project Progress

In an open source environment, the projects depend on voluntary contributions and, hence, the ability of a project to attract the interest of and contributions from developers is important for the project's success (Stewart et al., 2006). Also, many aspects of the project's development process are publicly visible through updates on the project's website or on repositories such as Sourceforge. Hence, Crowston et al (2006) reason that the data available about the development process can complement the measures used in studies on traditional software success. Thus, based on the OSS literature, two categories of predictors can be identified. The OSS license (Crowston et al., 2003; Stewart et al., 2006), operating system (Subramaniam et al., 2009), and programming language (Subramaniam et al., 2009) all represent the attributes of the OSS project itself, which have been shown to affect its success. In addition, the OSS license has been shown to have a significant impact on the project's success (Lerner and Tirole, 2005; Subramaniam et al, 2009). On the other hand, the developer interest (Krishnamurthy, 2002; Stewart et al., 2006) and end-user interest (Krishnamurthy, 2002; Parker & Van Alstyne, 2005; Stewart et al., 2006) are important to the success of and represent the OSS stakeholders' impacts on OSS projects. Thus, our research focuses on these two categories of predictors. Our hypotheses development begins with the explanation of the impact of OSS license.

OSS License: One of the main characteristics that differentiate various OSS licenses is the degree of restrictions imposed on the user to *re-distribute software derived or modified from OSS software* (Fershtman & Gandal, 2007). OSS license plays an important role in the success or failure of the project by impacting the interests of users and developers in the project (Subramaniam et al., 2009). For example, studies on OSS project performance find that users' interest in an OSS project and the project's activity levels are affected by the OSS license choice made by that project's administrators (Crowston et al., 2003; Stewart et al., 2006). The license is an important signal about the utility of an OSS project to the developers (Sen, Subramaniam, & Nelson, 2008-9). Lerner and Tirole (2005) propose three classes of OSS licenses based on the restrictiveness of redistribution rights (highly restrictive, restrictive, and unrestrictive). Other studies use the three levels of relative restrictiveness in their empirical studies on software licensing (Fershtman et al., 2007; Sen et al., 2008-9) and project success (Subramaniam et al., 2009).

Existing research on OSS project's *success* has found that restrictive licenses have an

adverse impact on user-interest in an OSS (e.g. Stewart et al., 2006). The license can also increase the complexity of working with the OSS product. This adverse impact could be attributed to resistance from organizations or individuals who prefer to retain the rights for reuse of the software code in a way that best serves their objectives. For example, software that includes any amount of GPL licensed (a Strong-Copyleft license which is highly restrictive) code has to be released under GPL license. Overall, the license choice can influence OSS project development timing at all stages and through various means. Potential project contributors, sponsors, advocates and users (both organizations and individuals) must make participation decisions, business judgments (and predictions) and schedule their timing. These business judgments must be carefully evaluated, weighed, and in some particularly complex cases (e.g. weak copy-left) with the consultation of intellectual property (IP) experts, attorneys and/or review boards. It is anticipated that fewer licensing restrictions are likely to attract greater stakeholder participation and cooperation, which in turn results in project managers to release a stable and functional product as quickly as possible. This leads to the following hypothesis:

> ***H1:*** *OSS projects that adopt less restrictive licenses will take relatively shorter duration to release a stable version of their software than do OSS projects that adopt more restrictive licenses.*

Operating System: The importance of the operating system for OSS is closely related to the Free Software Foundation launched by Stallman (2009), and the efforts of the Computer Science Research Group (CSRG) at the University of California at Berkeley to improve UNIX in the 1970s and 1980s. During the 1980s and early 1990s, open source software developers in several relatively isolated groups continued to use and improve the UNIX operating system as a voluntary community effort. The Internet and the user group USENET helped to coordinate their development efforts and much of the software developed by these different groups was integrated. As a result of this integration, complete environments could be built on top of UNIX using open source software. In short, developers using UNIX and Linux operating systems formed the initial core of the OSS community and these developers created various other applications, libraries, and utilities that complemented or supplemented the various flavors of UNIX and Linux in existence. It is only recently, that we have started to see other operating systems, such as Windows, being used in open source. Since most developers in the OSS community still work with UNIX/Linux operating system, we expect that OSS developed for UNIX or Linux operating systems will benefit from the ready availability of this operating system expertise, its' large installed base and compatibility with a wide range of products (and standards) and the associated positive network effects. This leads to the following hypothesis:

> ***H2:*** *OSS projects that develop software for UNIX/LINUX operating systems will take relatively shorter duration to release to a stable version of their software than do OSS projects that develop software for other operating systems.*

Programming Language: Unlike users of proprietary software, the users of open source software can make changes to the source code to fit their needs and hence the programming language of the software becomes important in determining the extent of participation in OSS projects. The dominance of C and C-like programming languages for OSS can be attributed to the role of C as the system implementation language for the nascent UNIX operating system (Ritchie, 1996). In fact, the UNIX kernel is written in C language and C is one of the preferred languages of OSS developers for codes that require portability, processing speed, real-time response needs, or tight coupling to the UNIX/Linux

kernel. Existing programs like parser generators or GUI builders that generate C code reduce the efforts required to code the rest of a small application using C. The availability of high-quality C compilers as open-source software over the Internet, including the best-known and most widely used Free Software Foundation's GNU C compiler, adds to the advantage of C programming language as a development platform. The C and C-like languages are still the preferred programming languages of software developers. We expect that open source software projects open to code written in C and C-like programming languages are likely to benefit through a large and diverse installed base of product compatibility, wide availability of subject matter experts (business and technical), code reuse, programming skills, software libraries, and other network effects of a well-developed, global development platform and experience. This leads to the following hypothesis:

> ***H3:*** *OSS projects that develop software using C and C-like programming languages will take relatively shorter duration to release to a stable version of their software than do OSS projects that use other programming languages.*

Developer Interest: Open source software is continually improved by feedback from the community and changes made in response to this feedback. Greater developer interest and participation in a project increases the speed with which features can be integrated into the software. More developers can also help to test earlier versions of the software and to identify and resolve bugs. Thus, a project's progress is enabled by its activity levels and is a sign of productive development community (Crowston et al., 2003; Stewart et al., 2002). One of the motivations for developers to participate in open source projects is to signal about their advanced programming skills to potential employers (Lerner et al., 2005) and to earn peer-recognition for these skills (Bonaccorsi et al., 2003). An active and successful project provides the developers with the increased visibility among potential employers and peers. Since projects reaching a stable stage quicker indicate more success among the open source community, we hypothesize that developer interest in an OSS project is also associated with faster project progress.

> ***H4:*** *OSS projects with greater developer interest will take relatively shorter duration to release a stable version of their software.*

User interest: One of the important roles provided by users in open source projects is as test subjects. Users who are interested in alternatives to their proprietary software may start using alpha or beta versions of the open source software and provide their feedback by way of identifying bugs or suggesting new features to be added. The more user interest a project attracts, it signals its utility to the OSS project managers and developers and leads to more project activity (Stewart et al., 2006). In order to permanently convert these users to their open source alternative, project administrators are motivated to provide a stable version sooner rather than later. Therefore, we hypothesize that user interest in an OSS project affects the project's progress.

> ***H5:*** *OSS projects with greater user interest will take relatively shorter duration to release a stable version of their software.*

The next section describes the statistical model and the data analysis.

Statistical Model, Data, and Estimation Results

Model: Since we are interested in investigating the determinants of the time taken to reach a certain status, we can use one of several survival models for analysis of the data. Survival models are used in studying the occurrence, progression and

timing of events. (Allison, 1995 page 1). Survival analysis techniques include life tables, Kaplan-Meier estimators, proportional hazards regressions, and competing risks models. *Survival models help to address two features of the data that cannot be handled with conventional statistical methods, such as logistical regression* (Allison, 1995 page 4). *These features are censoring (cases where the event has not yet occurred) and time-dependent covariates.* In the case of our study, survival models allow us to accommodate the fact that OSS projects reach stable status at different times, and some projects have yet to reach this stage (projects for which censored data is used). The Cox Proportional Hazard (PH) model, one of the semi parametric survival models, is considered a robust model since the coefficient estimates have good properties regardless of the actual shape of the baseline hazard function and in large samples the estimates are approximately unbiased and their sampling distribution is approximately normal (Allison, 1995 page 115). Thus, with the Cox PH model, using a minimum set of assumptions, we can obtain the primary information sought in this study.

However, a key assumption required for the Cox PH model is that the hazards for each predictor are proportional at all points in time (Allison, 1995) and that the hazards ratio does not change with time. We test the Cox PH model on our dataset first to estimate the constancy of hazards ratio of each predictors. As explained later in the "Model Estimation" section, some of the predictors violate the proportional hazards assumption, and these violations are equivalent to interactions between one or more covariates with time. Hence, we use the *extended Cox PH Model* (Allison, 1995), which is specified as follows.

$$h(t, X(t)) = h_o(t)\exp[\sum_{i=1}^{p1}\beta_i X_i + \sum_{j=1}^{p2}\beta_j X_j(t) \quad \text{(1b)}$$

where X_i *(i=1...p1)* are time-independent predictors, $X_j(t)$ *(j=1...p2)* are the interaction terms for the covariates that interact with time, $h_o(t)$ is the baseline hazard function, β_i and β_j are row vectors of the model coefficients. In the extended Cox model, if the coefficient of an interaction term is positive, then the effect of the related covariate increases linearly with time and if the coefficient is negative, the effect decreases with time. The corresponding coefficient of that covariate can be interpreted as the effects of the covariate on project progress at time zero (Allison, 1995). *A detailed explanation of our justifications for using the Extended Cox model and the interpretations of Cox regression coefficients are provided in Appendix A.*

Data: The data used in this study comes from the archives of Sourceforge.net [1], which maintains a large database of open source software projects. For each project, the database provides a description of the software, links for download and other project information, and a history of the project's releases. For our panel dataset, we used the monthly data of the projects. In our model, the event is an open source project releasing a stable version of the software (i.e., registering its software release status as Production/Stable or Mature in Sourceforge database). The date of entry of a project into our study is the date on which the project registered with Sourceforge. The dependent variable (Days_to_Stable) is the time between registering at Sourceforge and the date of Product/Stable or Mature release status. If the version released is not Production/Stable or Mature when our study completed (October 2005), the observation is right-censored. When studying events in survival analysis, the origin time is an important choice and researchers should

[1] Greg Madey, ed., The SourceForge Research Data Archive (SRDA). University of Notre Dame. <http://zerlot.cse.nd.edu/> [Last accessed 5/01/2006]

choose the origin that has the strongest effect on the hazard function. For example, when studying the treatment of a disease, the point of diagnosis of the disease may be a more useful and definitive origin point to understand the impact of the treatment options. A fundamental principle of open source projects is to allow users and developers, in addition to project initiators, to contribute by providing feedback, fixing bugs, designing new features, and refining code. Registering a project at Sourceforge, which is the leading registry of open source projects, helps project initiators to move their project to the open source arena. Hence, the point of origin in our study for the time to release a stable version begins when the project is registered on Sourceforge.

Since our data source is Sourceforge, we use the project status scale provided by Sourceforge (to be consistent across the projects). The Production/Stable and Mature status are the highest levels on the scale (we are ignoring the Inactive status). While the Sourceforge scale does not provide definitive descriptions of each level in the scale and project administrators may differ on the precise way to understand the scale, it is reasonable to expect the project administrators to agree that a Production/Stable or Mature software is one that has met the functional requirements set as the project goals. In general, projects in the Production/Stable and Mature status do not have bugs known to the project developers, perform as expected by the developers, and is ready to use "out of the box" by non-development users. Furthermore, one of the reasons for project administrators (who are in most cases themselves developers) to participate in open source projects is to enhance their reputation among peers and they have very little incentive to lie or exaggerate the project status.

To avoid the pitfalls of using a secondary data source such as Sourceforge.net (Howison et al., 2004), we did not use spiders or software to "screen-scrap" data from the Sourceforge.net website. Instead, we accessed data directly from a data warehouse, which is populated with Sourceforge data on a regular basis. The Sourceforge database contains information on more than 200,000 software projects. For the purpose of this study we consider only those projects which had registered between January 1999 and October 2005 (cut-off date) and for which complete information could be obtained. The number of such projects was 25,609. We found that 11,580 projects that had not released any files since their registration with Sourceforge. This lack of file release may indicate that the projects were inactive or were abandoned and we excluded these observations from our dataset.

In our dataset, there is a possibility that some of the projects had already developed a stable release before registering at Sourceforge (e.g., these projects may register with Sourceforge to get more exposure). When the event has occurred before the project enters our study, the observation is left-censored. To reduce the effects of such left-censored observations on our results, we exclude projects that had become stable within 100 days of registering with Sourceforge. There were 1006 projects excluded this way from our dataset. We also checked for outliers and found that while most projects had released their most recent file within 2000 days of registration at Sourceforge, there were eight projects that had released their most recent files after 3000 days. We excluded these eight projects from our study. There were no projects that had released their most recent files between 2000 and 3000 days since their registration. Our final sample size, thus, is 13,015.

The power (and the validity) of survival analysis is related to the number of events rather than the number of participants. Simulation studies have suggested that at least 10 events need to be observed for each covariate considered, and anything less could lead to biased regression coefficients (Peduzzi et al., 1995). In this

study, we have 10 covariates in the model (including the interactions of time-dependent variables and time), and therefore a sample size of 13,015 is considered adequate. There were 3,919 OSS projects in our sample that reported the development stage as Production/Stable or Mature (i.e., stable status). On an average it took 1006 days for the OSS projects to reach the stable status. The maximum number of days to reach stable status was 2000, while the minimum was 101 days. *Table 1* presents the measures of the independent variables collected from Sourceforge.net and *Table 2* presents the summary statistics of the sample of projects used in this study. While both the user interest and developer interest variables measure interest, their operationalization is different because of the different nature of these measures. Number of developers can go up or down every month. It is hard to get an exact count for total number of developers because the same developer may join a project and then leave it and then join back. Therefore, there could be common developers in each count. On the other hand, number of downloads only goes up. So we use the maximum of this number (in any month) which corresponds to the cumulative total number of downloads in a month.

Table 1 - Independent Variables and Descriptions/Measures

Independent Variable	Description/Measure
Developer Interest (i.e. Num-Developers)	Average number of developers who worked on the OSS project each month for which we have data.
User Interest (i.e. Downloads)	Total number of downloads of the OSS till time *t*
Strong-Copyleft	This measure equals 1 when the OSS is released under a Strong-Copyleft license such as GPL, and 0 otherwise.
Weak-Copyleft	This measure equals 1 when the OSS is released under a Weak-Copyleft license such as LGPL, and 0 otherwise
UNIX	This measure equals 1 when the OSS will work with various flavors of UNIX and Linux, and 0 otherwise.
CGroup	This measure equals 1 when the OSS or some component of the OSS was developed using C and C-like languages, and 0 otherwise.

As we can see from the summary statistics (Table 2), about 30% of projects are in stable status, and most of the projects release their software under Strong-Copyleft license (approximately 71%). The operating systems for 87% of the projects in our sample were the various flavors of UNIX and Linux, and 49% of the projects use C, C++, C# and/or Visual C programming languages. The distribution of the dependent variable *Days_to_Stable* for those OSS projects that achieved stable status is plotted in *Figure 2*.

Model Estimates: The acceptable goodness of fit of our model is indicated by the statistically significant chi-square value of the difference between the log likelihood (i.e., -2LL) measures of the null model and the proposed model (Hair et al., 2006). For our model χ^2 value is 1018.64 with 10 degrees of freedom (p=0.000), which implies that we can reject the null hypothesis that all effects of the independent variables on project progress are zero. Further, we use the Schoenfeld residuals to test for PH assumption and the results are shown in Table 3. As we see in *Table 3a*, the PH assumption is violated for

LN_Downloads (user interest), and UNIX (operating system), and also for the whole model (i.e. Global Test is significant at *p=0.05*). Therefore, the Extended Cox Model is appropriate for this data set. Finally, assuming a conservative threshold value of VIF<2 (The generally accepted threshold is VIF<10 (Hair et al., 2006) and the VIF in our study range from 1.01 to 1.62.) and Tolerance >0.3, we find that the model does not suffer from any significant multi-collinearity between independent variables of interest (*Table 3b*). The coefficient estimates and the corresponding hazard ratios for the model are shown in *Table 4.*

Table 2 - Descriptive Statistics

Total number of OSS projects in sample				13015
Number of OSS Project that have reached stable status				3919 (30.11%)
Censored observations (i.e. Projects that failed to reach stable status)				9096 (69.89%)
Predictors	**Min**	**Max**	**Mean**	**Std. Deviation**
Average Number of Developers	1	138	2.59	3.77
Downloads	1	8.60e+07	40732	870070
Predictors				**Approximate Proportion**
License is Strong-Copyleft (i.e. Strong_copyleft = 1)				71%
License is Weak-Copyleft (i.e. Weak_copyleft = 1)				13%
License is Non_Copyleft (i.e. Strong-Copyleft=0; Weak_Copyleft =0)				16%
Run on Various Flavors of UNIX/Linux (i.e. UNIX =1)				87%
Programming language used C and C-like (i.e. CGroup = 1)				49%

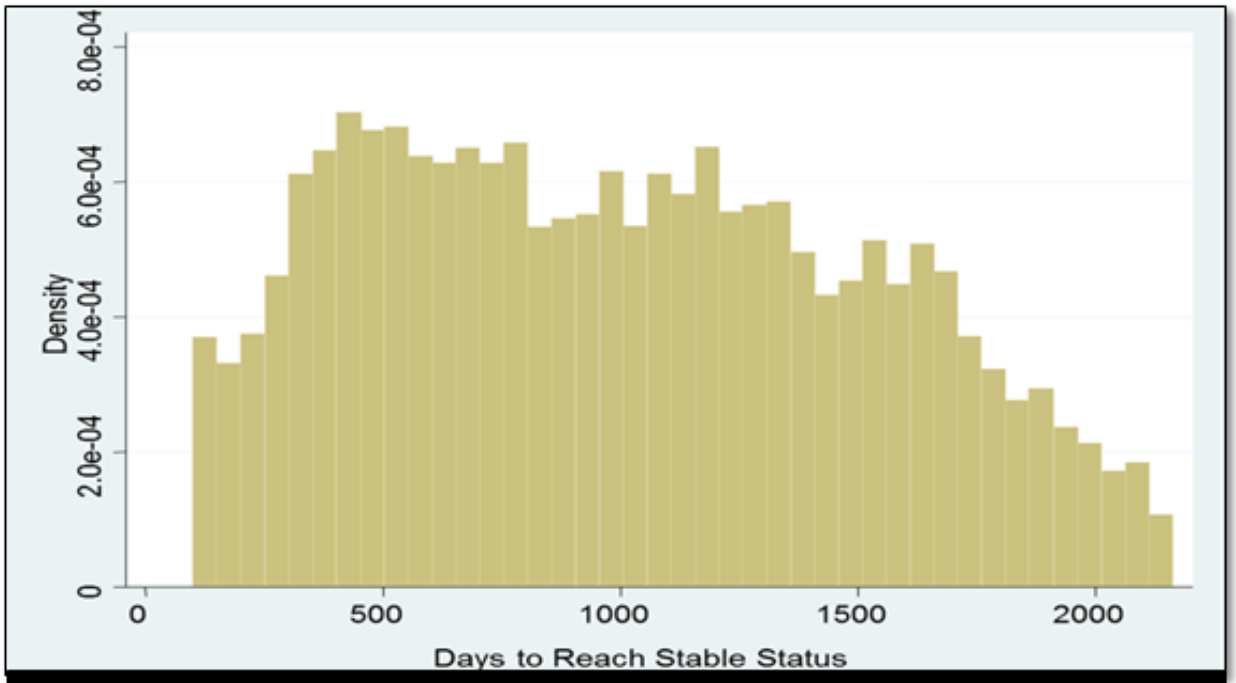

Figure 2: Distribution of Days to Reach Stable Status of OSS Projects

Table 3a - Test of PH Assumption				
	rho	**Chi2**	**df**	**Prob>Chi2**
Num_of_Developers	-0.020	3.34	1	0.0676
LN_Downloads	-0.091	34.70	1	0.0000
Strong_Copyleft	0.010	0.55	1	0.4590
Weak_Copyleft	-0.015	1.15	1	0.2839
UNIX	-0.032	4.75	1	0.0294
CGroup	0.021	2.27	1	0.1323
Developers*g(t)	0.017	2.47	1	0.1161
Download*g(t)	0.117	56.99	1	0.0000
Strong_Copyleft*g(t)	-0.018	1.64	1	0.2006
UNIX*g(t)	0.045	8.92	1	0.0086
Global Test		**110.50**	**10**	**0.0000**

Table 3b - Test of Multicolliearity				
PREDICTORS	**VIF**	**SQRT-VIF**	**Tolerance**	**R-Squared**
Num_of_Developers_mean	1.13	1.06	0.8850	0.1150
LN_Downloads	1.13	1.07	0.8813	0.1187
Strong	1.62	1.27	0.6163	0.3837
Weak	1.61	1.27	0.6194	0.3806
Unix	1.01	1.01	0.9880	0.0120
CGroup	1.02	1.01	0.9794	0.0206

Table 4 - Coefficient Estimates			
PREDICTORS	**Days-Stable**		
	Coefficients (β)	**Hazard Ratio**	**P>z**
Num_of_Developers	-0.347	0.707	0.000
LN_Downloads	-0.921	0.398	0.000
Strong-Copyleft	0.902	2.464	0.003
Weak-Copyleft	-0.159	0.853	0.007
UNIX	-2.300	0.100	0.000
CGroup	-0.265	0.767	0.000
Developers*g(t)	0.046	1.047	0.000
Download*g(t)	0.172	1.187	0.000
Strong_Copyleft*g(t)	-0.166	0.847	0.000
UNIX*g(t)	0.366	1.442	0.000
g(t)=ln(Days_to_Stable)			
Model χ^2 *(10) = 1018.64; Final model significance p<0.000*			

Appendix A provides a detailed explanation of the interpretations of the survival model coefficients. It also explains why the Cox model was chosen for our study. The significance of the coefficient of each predictor is used to assess the support for the relevant hypothesis. The hazard ratio (HR) for each predictor is computed using the coefficients of the predictor along with other interactive terms involving the predictor. An HR value greater than 1 for a predictor implies that an increase in the

value of the predictor will quicken the progress of the project towards stable status (positive impact). Alternatively, an HR value less than 1 implies that an increase in the value of the predictor will slow down the project progress (negative impact) and a HR value of 1 implies no effect of the predictor on the progress.

Table 5

Hypotheses	Result
H1: *OSS projects that adopt less restrictive licenses will take relatively shorter duration to release a stable version of their software than do OSS projects that adopt more restrictive licenses.*	*Weak-Copyleft Vs. Non-Copyleft:* **Supported**. *Strong-Copyleft Vs. Non-Copyleft:* **Supported** (Figure 3)
H2: *OSS projects that develop software for UNIX/LINUX operating systems will take relatively shorter duration to release to a stable version of their software than do OSS projects that develop software for non-UNIX/Linux operating systems.*	**Supported** (Figure 4)
H3: *OSS projects that develop software using C and C-like programming languages will take relatively shorter duration to release to a stable version of their software than do OSS projects that use other programming languages.*	Not Supported.
H4: *OSS projects with greater developer interest will take relatively shorter duration to release a stable version of their software.*	**Supported** (Figure 6)
H5: *OSS projects with greater user interest will take relatively shorter duration time to release a stable version of their software.*	**Supported** (Figure 7)

For independent predictors that interact with time (i.e., developer-interest, user-interest, license, and operating system), we plot the HR values for each predictor against the age of the project (i.e. *days since the project was registered at Sourceforge*) [2]. This allows us to trace the impact of the predictor on the progress of projects of various times since registration. Thus, the interpretation of the hypotheses test for these predictors should be made conditional on the time since registration of the project.

Discussion

The results of the hypotheses tests are summarized in *Table 5* and explained in detail in the following sections. The HR plots for predictors are shown in figures 2 through 5.

Impact of License on Project Progress (Hypothesis 1): Our results show that hypothesis H1 about license impacts is supported. The effect of any particular license choice on project progress is assessed on the basis of the hazard ratio (HR). The impact of Strong-Copleft license, compared to Non-Copyleft licenses, on Days_to_Stable is given by the expression

$$\exp(\beta_3 + \beta_9 \ln t) = \exp[0.902 - 0.166 * \ln(t)],$$

[2] To make our discussion easier to follow, we use the term "project age" to refer to the project's time since registration at Sourceforge.

where β_3 and β_9 are the coefficients of the license and license-time interaction variables respectively, and *t* ranges from 101-2000 days. As seen in Figure 3, the HR values are greater than 1 for projects whose time since registration is less than 229 days. This means that Strong-Copyleft license positively impacts a project's rate of progress if the project's time since registration is less than 229 days. For projects whose time since registration is greater than 229 days, having Strong-Copyleft license will slow the progress towards stable status (HR<1). Interestingly, Weak-Copyleft license has a negative impact on the time to reach stable status (HR<1) which means that projects with Weak-Copyleft licenses are likely to take longer to reach stable status compared to projects with Non-Copyleft licenses. This negative impact does not change with project's *time since registration* and hence we have not plotted the HR values in a graph.

The significant positive impact of Strong-Copyleft license during the early days of an OSS project could explain the widespread use of such licenses in OSS projects. Initially OSS project managers need to build support from the *open source community*, which favors Strong-Copyleft license such as GPL. In fact, the proponents of open source software advise project initiators to make their licenses GPL compatible[3]. Since more than 70% of the OSS projects registered at Sourceforge are released under GPL, it is reasonable to conclude that the open source community favors more restrictive licenses. The restrictive licenses could be favored by the OSS community for the following reasons: (a) The OSS community can benefit from the network effects generated by the large number of open source projects released under such a license (e.g. a choice of GPL license increases the likelihood that the software will be able to find another complementary software also released under GPL); (b) The OSS community uses the restrictive license to ensure that any derivatives of OSS are not released under commercial or proprietary licenses (Stallman, 2003). Therefore, project managers that release OSS under a strongly restrictive license should initially attract relatively more support from the open source community (including from project contributors), which in turn should reduce the time it takes for the project to release a stable version of the software.

This early support by the OSS community is evidently not without limitations, since Strong-Copyleft licensing (versus that of Non-Copyleft licensing) can have negative impact on longer-term projects (registered for more than 229 days). Over the longer-term (and consistent with Hypothesis 1), Non-Copyleft licensing can expedite project progress by reducing licensing complexity, opening the door to the entire software community (and beyond) and generating greater levels of organizational, commercial and individual interests. The finding that Weak-Copyleft (moderate licensing restrictions) has a consistent negative impact an OSS project's rate of progress also lends support regarding the notion that complexity in licensing can delay project progress. This helps in explaining the dominance of extreme licensing types of Stong-Copyleft and Non-Copyleft. However, when a project administrator utilizes a hybrid licensing arrangement (Weak-Copyleft), intellectual property (IP) and legal experts need to be consulted, business judgments must be carefully evaluated and weighed, ultimately causing delays in project progress.

Impact of Operating System on Project Progress (Hypothesis 2): Our results show that hypothesis H2 about the operating system impacts is supported. The effect of operating system choice on project progress is assessed on the basis of the hazard ratio (HR), which is given

[3] Wheeler, D.A. "Make Your Open Source Software GPL-Compatible. Or Else." Available at http://www.dwheeler.com/essays/gpl-compatible.html [Released 5/6/2002, revised 6/26/2013, last accessed 9/1/2013].

as

$$\exp(\beta_5 + \beta_{10} \ln t) = \exp[-2.3 + 0.366 * \ln(t)],$$

where β_5 and β_{10} are the coefficients of the operating system and operating system-time interaction variables respectively, and *t* ranges from 101-2000 days. The plot of HR in figure 4 shows a negative effect of Unix / Linux operating systems on project progress for the first 537 days, and then turns to a positive affect thereafter (since HR goes higher than 1.)

One explanation for this result is as follows. While there are historical reasons to suggest that the OSS community is more likely to favor UNIX/Linux operating systems (*Hypothesis 2*), the anticipated benefits of the choice of UNIX/Linux (e.g. availability of operating system expertise, large installed base and compatibility with a wide range of products, standards and the associated positive network effects) are delayed by 18 months. With 80% of OSS Projects registered at Sourceforge designated with Unix/Linux operating system, OSS stakeholders have a large selection of projects to participate in (or initiate on their own). Evidently the Unix/Linux designation is not distinctive enough to entice faster progress from the community during early phases of development. Once early milestones are achieved and project viability has been demonstrated, however, a bandwagon effect seems to take hold after 18 months.

Impact of Programming Language on Project Progress (Hypothesis 3): Since this predictor does not interact with time, we use the Kaplan-Meier failure estimates to understand its impact on the outcome. The plot of the failure estimates in Figure 5 shows that OSS projects' using C and C-like programming languages (i.e. C, C++, C#) progress towards releasing a stable software at a slower pace. This indicates that hypothesis H3 about programming language impacts is not supported.

One possible explanation could be that while experienced developers still favor C and C-like languages, newer developers lean towards more recent languages such as Java, Perl, and Php. Based on the statistics of top computer languages from Sourceforge, Java, Php, and Perl have been increasing in usage and together accounted for about 37% of usage in 2006.[4] The C and C-like languages contain several string functions that are prone to buffer over flow errors and lack features such as exception handling, function overloading, optional function arguments and garbage collection that most modern languages possess. One could argue that due to backward compatibility, C has not been updated to take advantage of increased memory and processor power to implement such things as automatic memory management. As a result of these limitations, projects that use C and C-like languages could have a more challenging time generating interests in the OSS community, attracting project sponsors, and achieving technical compatibility with new and emerging product lines. This could explain the slower pace of progress for projects using C and C-like languages in our study.

Impact of Developer Interest and User interest on Project Progress (Hypothesis 4 and Hypothesis 5): Our results show that hypotheses H4 and H5 are supported. The effects of developer-interest on project progress is assessed on the basis of the hazard ratio (HR), given as

$$\exp(\beta_1 + \beta_7 \ln t) = \exp[-0.347 + 0.046 * \ln(t)]$$

, where β_1 and β_7 are the coefficients of the developer interest and developer interest-time-time interaction variables respectively, and *t* ranges from 101-2000 days. As seen in figure 6, developer-interests have a positive impact on progress only for projects with more than 1889 days since registration at Sourceforge.

The effect of user-interest on project progress is assessed on the basis of the hazard ratio (HR), which is given

[4] http://www.cs.berkeley.edu/~flab/languages.html [Last accessed 05/07/08]

by

$$\exp(\beta_2 + \beta_8 \ln t) = \exp[-0.901 + 0.172 * \ln(t)]$$

, where β_2 and β_8 are the coefficients of the user-interest and user interest-time interaction variables respectively, and t ranges from 101-2000 days. As seen in figure 7, user interest has a positive effect for projects with more than 189 days since registration.

The level of user-interest in an OSS project has an earlier (and greater) impact on its progress towards a stable release than does the level of developer-interest. The presence of more developers may result in continuing add-ons or modifications to the software, thus delaying the project from releasing a stable version. Thus, for most OSS projects that are not very large, the project administrators could implement policies to control the excess participation of developers. Some of these policies could include accepting only changes that add significant value to the software or forming a smaller group of developers who can make quick decisions on the software specifications. More end-users, on the other hand, may motivate OSS project managers to release a usable product quickly and build a community loyal to the project.

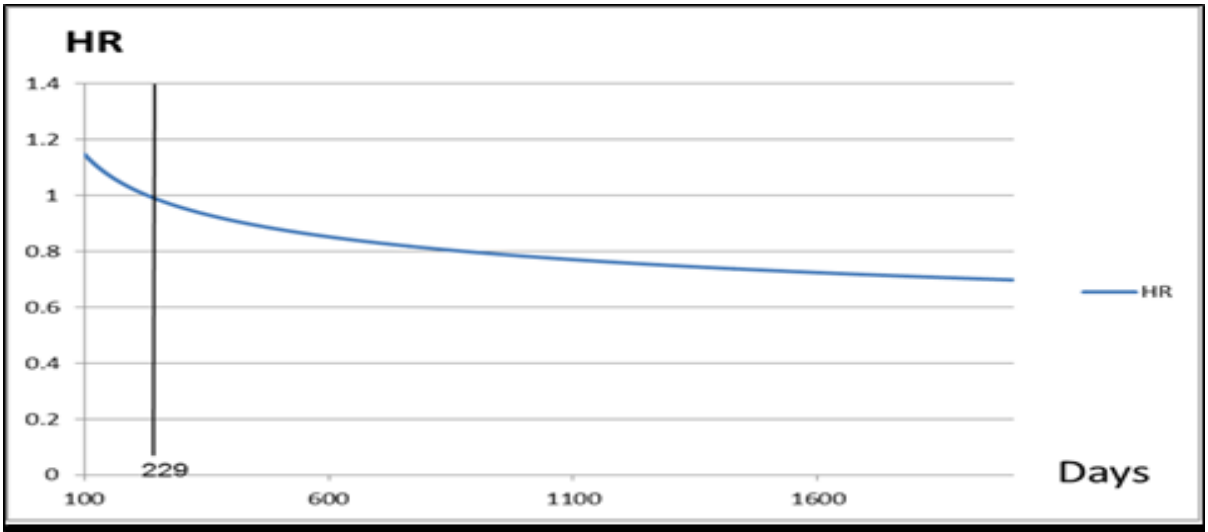

Figure 3: Plot of HR for Strong-Copyleft License

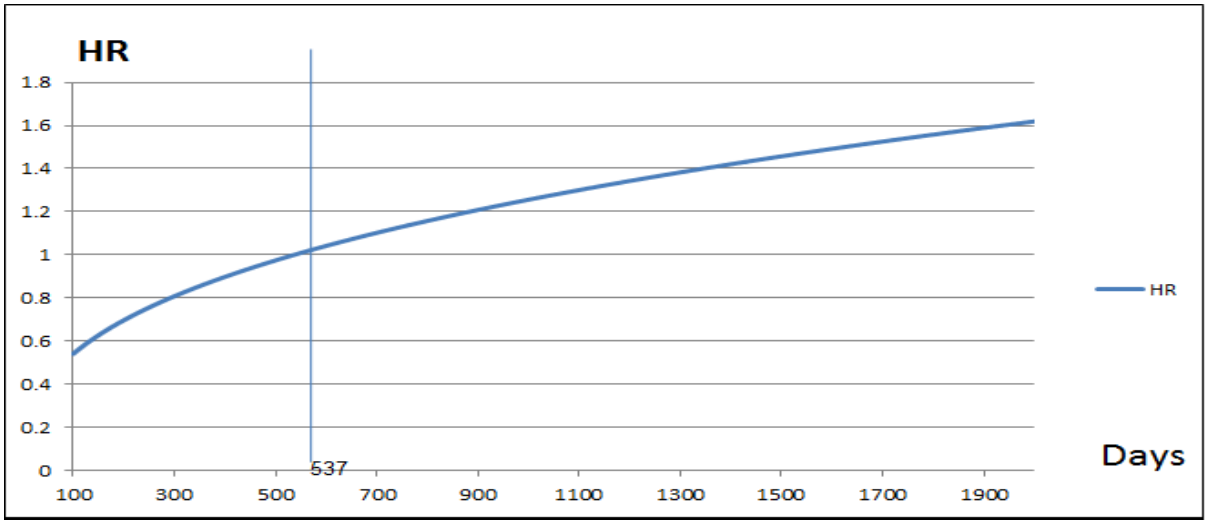

Figure 4: Plot of HR for OSS Projects Developed for Unix/Linux

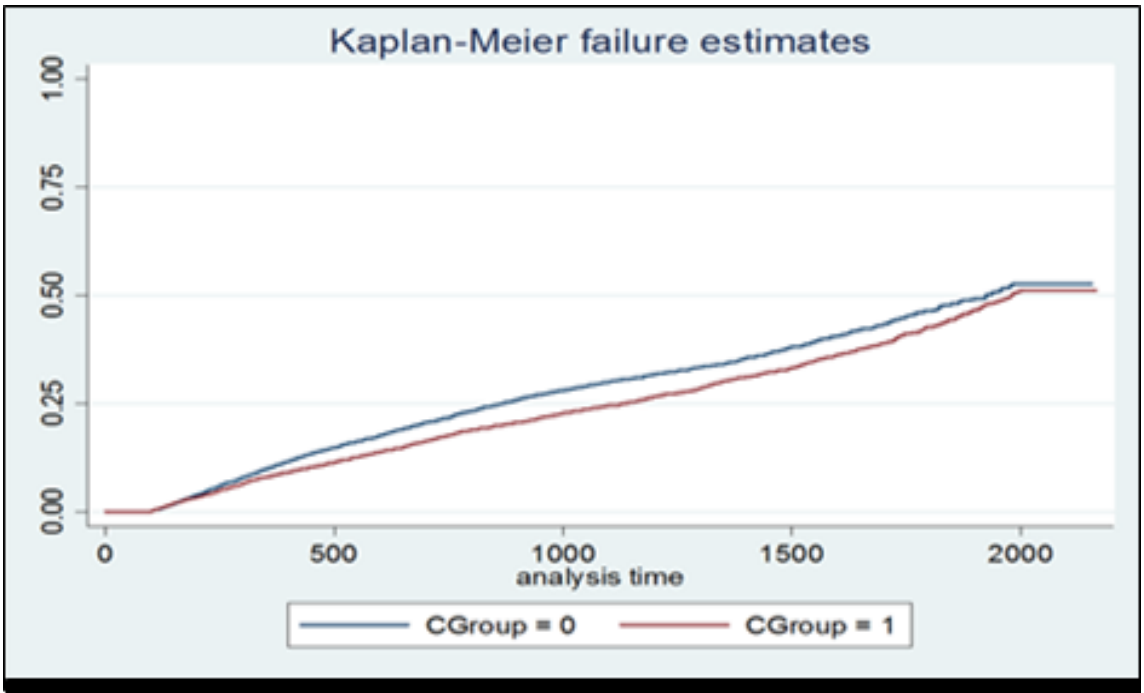

Figure 5: Compared to Other Programming Languages, OSS Developed in C Group of Languages Reach Stable Status Later

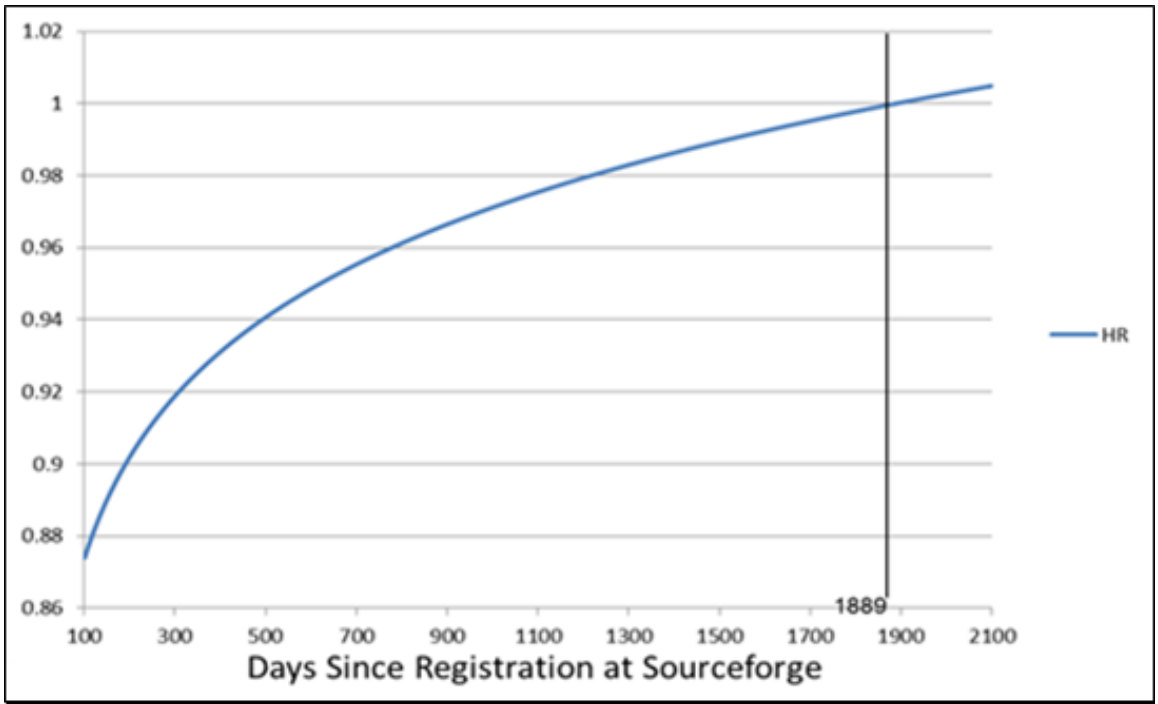

Figure 6: Plot of HR for Developer Interest

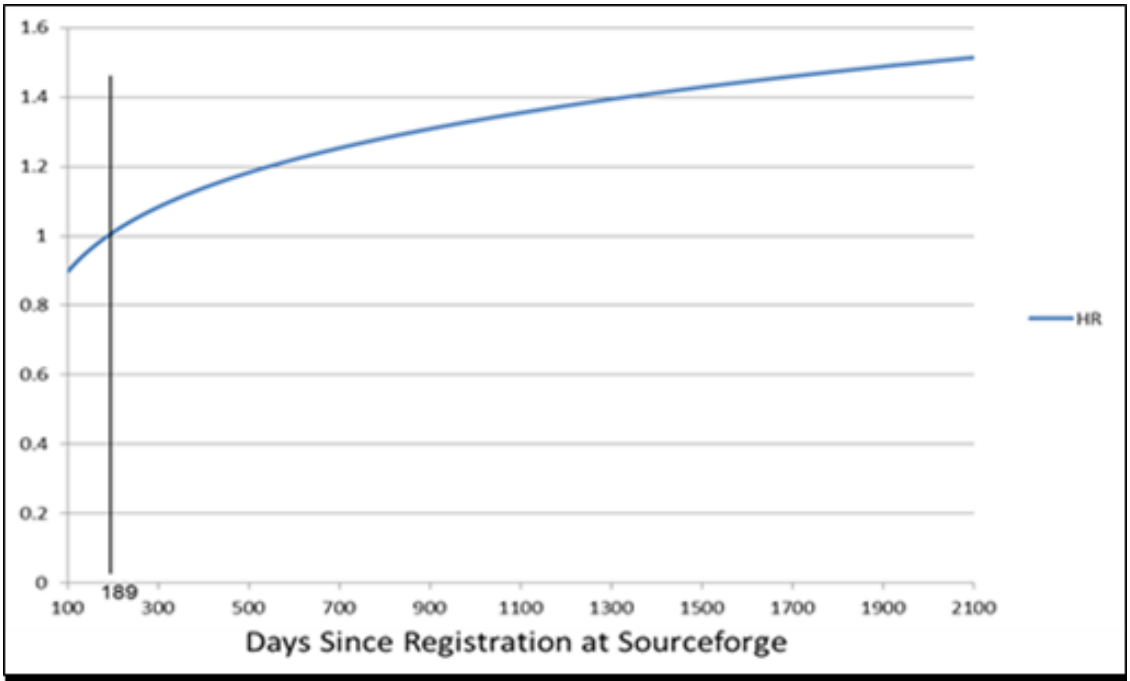

Figure 7: Plot of HR for User Interest

Conclusion

Collectively, the findings reveal an OSS community at a cross-roads - between its rich history of a close-knit, developer centric community on the one hand, and the growing influence of a broader-base, end-user orientated community on the other. Specifically, user-interests were found to positively influence OSS project progress four and one-half years earlier than developer interests. The use of conventional programming languages such as C and C-like languages negatively impact project progress, as does the use of Unix / Linux based operating systems during the first one and one-half years of a project's duration. Also, the OSS community's preference for Strong-Copyleft licensing does positively influence OSS progress on shorter-term projects (less than 8 months in duration) and the broader-based, less restrictive Non-Copyleft licensing has a positive influence on OSS progress thereafter.

One of the limitations of our paper is that the data can be considered as old. However, the variables used in our study are neither subjective nor contextual and have been shown to consistently relate to project success. Hence, we believe that our results will hold even for more recent data. Our paper leaves some issues unaddressed, which could be investigated in future research. The indicator variables in our model for project progress do not include characteristics specific to project developers. The impacts of the developers' characteristics on project progress could be very insightful, especially if their simultaneous impacts on the choice of end-user license, programming language, and operating system are considered in the analysis.

The results of this paper can be interpreted as a tool to understand the importance of the OSS project's characteristics in determining its contribution to the open source community, as measured by its ability to provide a stable product. This research also partially explains the prevalence of restrictive Strong-Copyleft licenses, such as GPL, in open source projects. OSS projects that use more

recent software tools, such as the Windows operating system or Java programming language are more likely to release a stable product faster than projects that use the traditional hallmarks of open source – UNIX operating system and C programming language. Other researchers can further investigate the inter-relationships among the predictors identified in our study and develop a more comprehensive model of OSS project progress. From a practitioner perspective, our results can help OSS project managers to understand which OSS project characteristics can be controlled in order to meet the project goals.

Acknowledgements

The data for this study was made available from the Sourceforge Research Data Archive (SRDA) maintained by Prof. Greg Madey at the University of Notre Dame http://zerlot.cse.nd.edu/

References

Allison, P.D. (1995). "Survival Analysis Using SAS: A Practical Guide," Cary, NC: SAS Institute Inc.

Bardhan, I.R., Kauffman, R.J. and Naranpanawe, S. (2010). "IT project portfolio optimization: A risk management approach to software development governance", *IBM Journal Research & Development*, 54(2), March / April 2010, pp1-18.

Bonaccorsi, A. and Rossi, C. (2003). "Why Open Source Software Can Succeed?," *Research Policy,* (32:7), 2003, pp 1243.

Cox, D.R. "Regression Models and Life Tables," (1972). *Journal of Royal Statistical Society,* Series B, (34), 1972, pp 187-220.

Crowston, K., Howison, J., and Annabi, H. (2006). "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice,* (11:2), 2006, pp 123-148.

Crowston, K., Annabi, H., and Howison, J. (2003). "Defining Open Source Software Project Success." In *Proceedings of the 24th International Conference on Information Systems,* Seattle, WA. December 2003.

Crowston, K. and Scozzi, B. (2002). "Open source software projects as virtual organizations: Competency Rallying for Software Development," In *IEEE Proceedings Software*, (149:1), 2002, pp 3-17.

DeLone, W. and McLean, E.R. (2003). "The DeLone and McLean Model of Information Systems Success: A Ten-Year Update," *Journal of Management Information Systems*, (19:4), Spring 2003, pp 9–30.

.

Driver, M, "Key Issues for Open-Source Software, 2010", Gartner Research #G00175310, April 26, 2010.

Ewusi-Mensah, K. (1997). "Critical Issues in Abandoned Information Systems Development Projects," *Communications of the ACM,* (40:9), pp 74-80.

Fershtman, C. and Gandal, N. (2007). "Open Source Software: Motivation and Restrictive Licensing," *International Economics and Economic Policy,* (4:2), pp 209-225.

Fitzgerald, B. (2006). "The Transformation of Open Source Software", *MIS Quarterly*, 30(3), 587-598.

Gorton, I., and Liu, A. (2002). "Software Component Quality Assessment in Practice: Successes and Practical Impediments," In *Proceedings of the 24th International Conference on Software Engineering*, IEEE Computer Society, pp 555-558.

Grewal, R., Lilien, G.L., and Mallapragada, G. (2006). "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science,* 52(7), July 2006, pp 1043-1056.

Hair, J. F. Jr. et al. *"Multivariate Data Analysis"* (Sixth Edition), Pearson Prentice Hall, 2006.

Hann, I., Robert, J., and Slaughter, S.A. (2004). "Why developers participate in open source software projects: an empirical investigation" In *Proceedings of the 25th International Conference on Information Systems*, 13-15th December 2004, pp 821-830.

Hoffer, J.A., George, J.F., and Valacich, J.S. (2008). *"Modern Systems Analysis and Design"* (Fifth Edition), Pearson Prentice Hall, 2008.

Howison, J. and Crowston, K. "The Perils and Pitfalls of Mining Sourceforge," (2004). In *Proceedings of Mining Software Repositories Workshop, International Conference on Software Engineering, Edinburgh, Scotland*, May 25 2004.

Kogut, B. and Metiu, A. (2001). "Open Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy,* (17:2), pp. 248-264.

Krishnamurthy, S. (2002). "Cave or community? An empirical investigation of 100 mature open source projects" *First Monday*, (7:6), 2002.

Lerner, J. and Tirole, J. (2005). "The Scope of Open Source Licensing," *Journal of Law, Economics, and Organization*, (21:1), April 2005, pp 20-56.

Martin, B. (1998). "Information Liberation," Freedom Press: London (UK), p. 29-56. 1998. [Available at www.uow.edu.au/arts/sts/bmartin/pubs/98il/il03.html Last accessed 5/01/2006].

Mockus, A., Fielding, R.T., and Herbsleb, J.D. (2002). "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology,* (11:3), pp 309-346.

Nelson, M., Sen, R., and Subramaniam, C. (2006). "Understanding Open Source Software Development- A Research Classification Framework," *Communications of AIS,* (17:12), pp 266-287.

Parker, G. and Van Alstyne, M. "Innovation through optimal licensing in free markets and free software." Available at http://ssrn.com/abstract=639165 (Last accessed 10/20/2007].

Peduzzi, P., Concato, J., Feinstein, A.R., and Holford, T.R. (1995). "Importance of events per independent variable in proportional hazards regression analysis II: accuracy and precision of regression estimates," *Journal of Clinical Epidemiology,* (48), 1995, pp 1503−1510.

Ritchie, D.M. (1996). *"The development of the C Language,"* in Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. (Eds) History of Programming Languages-II, ACM Press (New York) and Addison-Wesley (Reading, Mass).

Sen, R. (2007). "A Strategic Analysis of Competition between Open Source

and Proprietary Software Applications," *Journal of Management Information Systems*, (24:1), Summer 2007, pp 233-258.

Sen, R., Subramaniam, C., and Nelson, M.L. (2008-9). "Determinants of the Choice of Open Source Software License," *Journal of Management Information Systems*, (25:3), Winter 2008-9, pp. 207-239.

Stallman, R.M. (2009). "Why "Open Source" Misses the Point of Free Software", *Communications of the ACM*, 52:6, June 2009, pp 31-33.

Stallman, R.M. (2003). "Copyleft: Pragmatic Idealism," In *Free Software, Free Society: The Selected Essays of Richard M. Stallman.* Available at http://www.gnu.org/philosophy/pragmatic.html [Last Accessed on 05/01/06].

Stewart, K. J., and Ammeter, T. (2002). "An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects," *Twenty-Third International Conference on Information Systems*, pp 853-857.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. (2006). "Impact of license choice and organizational sponsorship on success in open source software development projects," *Information System Research,* (172), pp 126-144.

Stewart, K.J., and Gosain, S. (2006). "The impact of ideology on effectiveness in open source software development teams," *MIS Quarterly*, (30:2), pp 291-314.

Subramaniam, C., Sen, R., and Nelson, M.L. (2009). "Determinants of Open Source Software Project Success: A Longitudinal Study," *Decision Support Systems*, (46), pp 576-585.

Appendix A

In this appendix, we explain why Cox regression model is more suitable to our study than other multivariate models, including structural equation models. We also explain how the Cox regression coefficients are interpreted.

The following are the reasons for using Cox regression for our study.

1. **Focus of the Study**- Cox regression is designed for analysis of *time until an event or time between events,* and out paper studies time until an event.
2. **Censored Data**: Censored observations occur in all "time to event" data unless the data are historical, with all data present for all observations. Traditional regression methods would require either dropping censored cases, thereby risking sample selection bias, or treating censored cases the same as those for whom the event occurred in the final time period of observation, which will also bias the coefficient estimates. Unlike these regression methods that use maximum likelihood estimation of parameters, Cox regression uses partial likelihood methods, which do not assume uncensored data. In Cox regression, the coefficient estimates are based only on the uncensored cases, but all cases are used when estimating the baseline hazard function. Thus Cox regression uses all available information and is considered a full information method. Since we have censored data in this study, we believe that Cox regression is a more suitable method.
3. **Time Varying Independent Variables***:* Downloads and Developer-Interest could be time varying. Time varying independent variables such as these can be handled in Cox regression but not in traditional regression.

Interpreting the model coefficients: The interpretation of the coefficient estimates is made easier by using the Hazard ratio (HR), also called the odds ratio. The hazard ratio is the probability of the event occurring in time t + 1, given survival to time t (i.e. given that the event has not occurred till time t). A hazard ratio of 1 indicates the variables in the model have no effect on time to event for the status variable. For hazard ratio below 1, the greater the covariate, the less the odds of the event occurring (increasing predicted survival times). For hazard ratio above 1, the greater the covariate, the higher the odds of the event occurring. For instance, if a covariate is Weak-Copyleft (0, 1) with 1 being Weak-Copyleft licensed OSS, and if the hazard ratio is 1.1, and if the event is Status=Stable, then the risk of reaching stability is 1.1 times greater for OSS with Weak-Copyleft than for OSS with other licenses (Weak-Copyleft=0), controlling for any other covariates in the model.

For independent predictors that interact with time (i.e., developer-interest, user-interest, license, and operating system), we plot the HR values for each predictor against the age of the project (i.e. *days since the project was registered at Sourceforge*) [5] . This allows us to trace the impact of the predictor on the progress of projects of various times since registration. Thus, the interpretation of the hypotheses test for these predictors should be made conditional on the time since registration of the project.

About the Authors

Ravi Sen is an Associate Professor in the department of Information and Operations Management at the Mays Business School, Texas A&M University. He received his Ph.D. in Business Administration from the University of Illinois at Urbana–Champaign in 2003. His research interests include economics of electronic commerce, open source software, and software security. He has published in the *Journal of Management Information Systems, Decision Support Systems, International Journal of Electronic Commerce, Communications of the AIS, Electronic Markets, Journal of Electronic Commerce Research*, and others.

Matthew L. Nelson is an Associate Professor in the Department of Accounting and Business Information Systems. He holds a PhD from the University of Illinois at Urbana–Champaign. His research interests include information technology valuation, open source software, and IT standards. He has published in *Journal of Management Information Systems, Decision Support Systems, Information and Management, Communications of the Association for Information Systems, Electronic Markets, Mathematical and Computer Modeling* and others.

Chandrasekar Subramaniam is an Associate Professor in the Department of Business Information Systems and Operations Management at the Belk College of Business, University of North Carolina at Charlotte. He received his PhD from the University of Illinois at Urbana–Champaign. His current research interests include data analytics, e-business, value of information technology, interorganizational systems, open source software, and IT security. He has published in several MIS journals, including *Journal of Management Information Systems, Decision Support Systems, International Journal of Electronic Commerce, Communications of the AIS,* and *Information Systems Frontiers.*

[5] To make our discussion easier to follow, we use the term “project age” to refer to the project’s time since registration at Sourceforge.