

# **An Empirical Investigation of Code Contribution, Communication Participation, and Release Strategy in Open Source Software Development: A Conditional Hazard Model Approach**

Param Vir Singh • Ming Fan • Yong Tan

Information Systems and Operations Management  
University of Washington, Seattle, Washington 98195  
{psidhu • mfan • ytan}@u.washington.edu

January 4, 2007

## **Abstract**

This study uses data from over 200 open source software (OSS) projects hosted at Sourceforge to empirically examine how OSS development characteristics affect project progress. Consistent with prior studies, we find that core developers (i.e., top 20 % of the most contributing developers) develop about 80% of the code. We also find that a group of developers that is about 50% larger than the core group is responsible for 80% of the communication. On average, the top 30% developers contribute about 81% of the messages. It is observed that projects with a clearly identified core group of developers conducting most of the coding are more efficient for project progress. We also find that highly participative peripheral group in communication positively affects project progress. In addition, high level of communication interactivity, measured by the number of email threads and average thread depth, enables a project to progress well. Our results also show that projects that draw upon the larger OSS community for beta testing progress faster. We find that the projects that do not progress well are associated with the following major factors: (i) lack of a core developer group, (ii) dormant peripheral group in communication, (iii) low communication interactivity exemplified by the lack of depth in threaded communications, and (iv) over-dependence on internal community for beta testing.

**Key words:** *Open source software, software development time, software engineering, coordination, duration time modeling*

## 1. Introduction

OSS phenomenon derives its strength by harnessing the free-of-charge large-scale R&D resources through the formation of developer communities [18]. These communities differ in their level of participations in coding and communication. For instance, some communities have a very close knit group of core developers and a highly participative peripheral group. In contrast, another community may have a diffused core, i.e., a clear distinction between a core and a peripheral developer may be absent. Further, some communities draw upon the larger OSS community for beta testing, compared to others which prefer to fix bugs in-house (internal community) before releasing the software to public. These strategies (coding, communication, and release) have their strengths and weaknesses which have implications for the progress of a project. In this study, we develop measures that can be used to quantify these strategies and investigate the impact of code contribution, communication participation, and release strategies on the progress of a project.

OSS proponents have argued that the strengths of the OSS phenomenon are associated with the *bazaar* model of full participation or participation of abundance from large number of developers and users [18], [23]. This argument is based on the logic that a highly participative community may lead to richer discussion, better flow of ideas, efficient code development, faster bug finding and fixing, and, hence, faster and efficient project growth [2], [18], [23]. However, contrary to the bazaar view of OSS projects, many empirical studies have found that only a small number of developers contribute a large percentage of code and discussion in OSS projects [4], [11], [12], [13]. Moreover, there is no consensus on the benefits of the *bazaar* model. Cox [2], a senior developer at Linux, argues that the *bazaar* model may turn into a “town council” where equally participative community produces only noise and hence no output. He further claims that

such a bazaar-town council model may slow down the evolution of a project or even act as a recipe for project failure. Such conflicting arguments on the effects of member participation raise the question whether more equal participations are better or worse for OSS projects.

While OSS proponents have not distinguished between coding and communication while discussing participation strategies, researchers on participation in OSS projects have largely focused on code contributions. However, in addition to code contributions, communication and discussion play an important role in OSS development. In contrast to the view that the main part of OSS development is in writing, debugging, and rewriting codes [18], developers are constantly involved in high level interactions with each other in learning, knowledge sharing, critiquing, and redesigning [6]. Many times, the primary difficulty in OSS development is not the implementation of a solution; rather, it is in deciding which of various possibilities is the most appropriate solution [13]. This requires constant communication between OSS members. In this study, we investigate the impacts of both code and communication contributions on the progress of an OSS project.

Several measures of success/progress for an OSS project have been proposed in literature: number of downloads, page views, number of users, number of developers, re-use of code, bug-fix turn around time, and development time for the first stable release [3], [7], [16]. Out of these measures, the development time for the first stable release by a project, when properly adjusted by the project and team sizes, appears to be a legitimate measure that can be used to compare the impact of code and communication contributions on the progress across projects [3], [17]. Producing a stable release is often considered a sign of project takeoff and initial project success [3], [13]. Besides, our data shows that most OSS teams have problems in achieving production stability. For instance, about 82% of the more than 100,000 projects hosted at Sourceforge have

not released stable versions by July 2006. Using this success measure also allows us to examine the various release strategies followed by project teams. A project may release a buggy product to access the large pool of beta testers from the outside OSS community. On the contrary, a project may first fix all the bugs in-house and release only the stable product.

The following are the salient features of this study:

- Studies on OSS have mainly examined a small number of large, well known, and successful projects as case studies, e.g. in [4], [13]. These studies are important to identify major issues in OSS development. It will be interesting to see whether findings in prior case studies will hold for a large number of relatively small OSS projects. We collected data from 205 OSS projects hosted at Sourceforge. Our sample includes varieties of projects, ranging from relatively large to small projects, and projects that are successful and projects that do not progress that well.
- We extend prior studies on code contribution strategies, and develop measures on the level of inequalities in both code contributions and communication participations, as well as the level of communication interactivity in OSS projects. We employ quantitative methods to examine the effects of these factors on the development time of the first stable release.
- Ordinary least squares (OLS) regression is inappropriate to study the impact of these factors on the development time due to the problems of censoring and omitted variable bias. The sample that we have suffers from right censoring as it contains some projects which have not released a stable version by the data collection time. Excluding these projects may introduce a sample selection bias. However, one needs to control for right censoring if these projects are included in the sample. The development time is also likely to be influenced by unobserved factors such as inherent complexity of the software or

ability of the developers involved, which may cause serious endogeneity problems leading to spurious correlations [20]. We propose a hazard function model to address these issues. Hazard models of event durations and timing have found applications in many areas such as the length of survival after medical treatments [10], labor strike durations, and inter-purchase timing [20]. Hazard models are superior to the general regression models to study duration data because hazard functions can handle censoring and control for unobserved heterogeneity (omitted variable bias), thus ensuring consistent and unbiased estimates.

The rest of the paper is organized as follows. Section 2 discusses related work. We describe our research design and data collection in Section 3. The details of the hazard model are presented in Section 4. Section 5 summarizes the results of our study and discusses the implications. We conclude in Section 6.

## **2. Related Work**

There is a growing body of research on OSS development. OSS projects are based on the voluntary contributions of a large number of developers [18]. However, much of the development activity is often realized by a small number of developers [11], [14]. Mockus et al. [13] found that the top 15 developers contributed about 83% of the modification requests and 88% of the added lines in the Apache project. Similarly, in a study of the FreeBSD project, Dinh-Trong and Bieman [4] found that a group of the top developers contributed about 80% of new functionality.

Researchers are also interested in the software development and coordination process in OSS projects. Studies have examined the effectiveness of the lean coordination tools such as mailing lists and CVS [13], [25]. OSS developers often work in parallels and experiment with different routes to a resolution. Coordination in OSS community is quite difficult from that in

formal organizations. Developers rely heavily on communication, especially mailing lists, to discuss about project progress, identify work to be done, assign development tasks, and coordinate releases [13], [23]. There are diverse communication and coordination patterns in OSS communities. For example, major decisions at Apache project use email voting and any developers who contribute to Apache can vote on any issue by sending email to the mailing list [5]. In contrast, Larry Wall, the originator of Perl, developed a delegated decision-making structure [23].

Communications of OSS projects also play important roles in knowledge sharing and collective innovation [12]. In contrast to the view that the major OSS development is just about writing and debugging code, developers are constantly involved in high level interactions with each other in learning, sharing knowledge, and redesigning [6]. Many times, analyzing and discussing various solution options are more important than implementing one solution [13]. Thus, communication plays a critical role in OSS development. Singh et al. [19] found that developers learn from their peers through interactions as well as through prior code contributions. Such knowledge sharing has long term positive impact on the future code contributions of the developers involved. However, some believe that not all the communications are valuable to the development process [21].

Prior studies have also looked at OSS release strategies. Many OSS projects adopt “release often and release early” strategy [18], which takes advantage of the feedback mechanisms from the large development base of OSS. This approach differs significantly from traditional software engineering methodology in commercial software products.

Paulson et al. [16] have compared the performance between open-source and closed-source software products in terms of project growth, simplicity, and defects. They did not find evidence

that OSS projects grow more quickly than closed-source ones. Crowston et al. [3] suggest different measures on OSS project success. Project progress, especially stable release of a product, is an important indicator of project success. Rajagopalan and Bayus [17] use project takeoff, which is identified through download patterns, to measure project progress. In this study, we extend the literature on OSS development and mainly examine how code contribution, communication participation, and release strategies may affect the progress of a project.

### **3. Research Design and Data Collection**

Data for this study were collected from Sourceforge.net. Sourceforge is the world's largest OSS project repository with more than 1,000,000 registered users and more than 100,000 projects. It provides a good sample of the OSS community to study the underlying dynamics. Our study considers all the projects that were registered before July 31, 2004 and fall under the category "Multimedia" at Sourceforge. We used web agents to collect data from Sourceforge. The complete mailing list archives, Concurrent Versioning Systems (CVS) log files, project "release notes", "news", and download statistics were downloaded for each project.

Mailing lists at Sourceforge provide communication data for this study. Projects hosted at Sourceforge use mailing lists for developers to communicate with each other. Usually, projects at Sourceforge have their own websites. It is possible that the projects also use mailing lists on their own websites for developers' communication. Due to the difficulty in matching the identities of the developers at the two places (Sourceforge and the project's homepage), we only include in this study those projects that solely use Sourceforge mailing lists as their communication tool. To identify the projects that only use Sourceforge mailing lists for communication, we checked the homepages of the studied projects, which provide information on the communication source for

the developers. In processing mailing lists data, we retrieve the sender's id, name, sending date, and the thread-id.

The code contribution data for this study is collected from the project CVS repositories at Sourceforge. The CVS repository contains the current state of the project as well as the previous versions of the code. The CVS repositories contain information about the code changes (number of lines added or subtracted), date of change, and the identifier of the developer who made the changes. We extract those data from CVS entries. In some projects, only a selected few developers are given the power to commit to the CVS. Under those situations, the CVS logs may be biased in tracking development contributions. Thus, we only consider those projects where all the registered developers have been granted the CVS commit power. In the final sample, we have 205 projects. Table 1 provides a description of the sample. As the descriptive statistics show, our sample includes a wide variety of projects.

**Table 1.** Distributions of the Sample Projects (%)

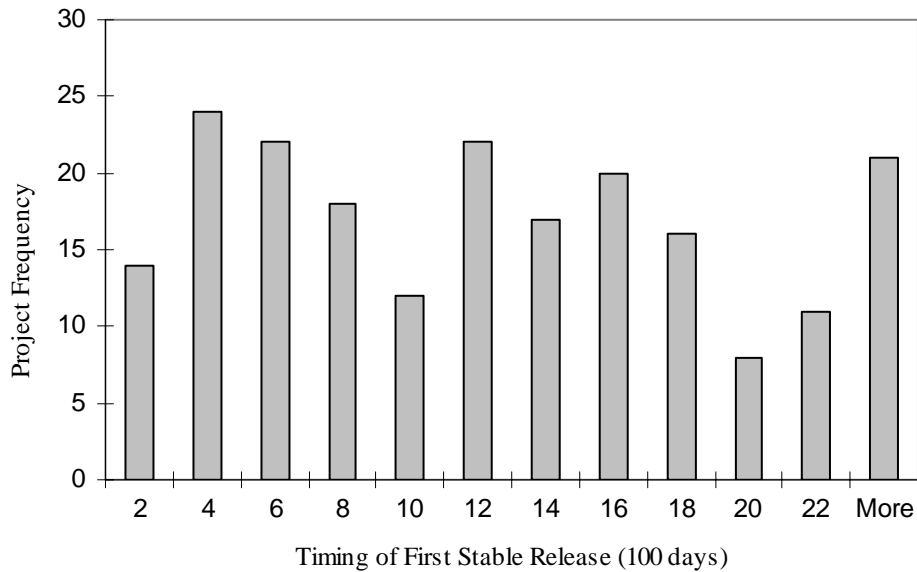
Translation		Topic		Environment		Operating System (OS)	
English	90.8	Games, Entertainment	15.2	Web-Based	24.9	Windows	44.70
German	13.4	Database	16.1	Win32	23.5	POSIX	67.74
Spanish	8.3	Software Development	31.3	X11	31.3	OS Independent	40.55
Italian	6.0	System	51.2	Non-Interactive	12.4	OS Distribution Specific	14.75
Russian	6.0	Internet	25.8	Cocoa	4.6	Other	6.91
French	12.9	Multimedia	100	Gnome	9.2		
Chinese	5.5	Office/Business	7.4	KDE	3.7		
Japanese	5.1	Scientific, Engineering	22.1	Handheld	1.4		
Other	39.6	Desktop Environment	6.5	Other	26.7		
		Formats and Protocols	6.0				
		Communications	22.6				
		Other	16.6				

We here provide descriptions for the measures that are used for this study:

**Software Stable Release.** Release strategies vary across OSS projects. However, most of the versions released by an OSS project can be classified as functional enhancement or bug-fixing releases. A project may release a significantly functionally enhanced package and follow up with several quick bug-fix releases. A project team may decide not to release a functionally enhanced package until they have ironed out the bugs internally. A Project team may also release a package with minimal functionality with/without rigorous internal testing. These varying possibilities make it difficult for the stable release to be comparable across projects. OSS proponents even argue that an OSS is always in the beta state as continuous updating goes on raising questions about the stability of a package. This makes identification of the stable release of a project somewhat tricky. To overcome these problems, we focus on the stability as well as functionality of the releases.

The information that helps identify the stable release of the projects comes from two sources: user download data and the “release notes” or “news” corresponding to each package released by the project since its inception at Sourceforge. First, following the literature on product takeoff [1], [8], [17], we require that the stable release should bring in enough users, i.e. the number of downloads should have a significant jump, compared to any prior releases, if any. Second, we mine project “release notes” and “news”, which provide additional information on the functionality, bug fixing and the stability of the associated package. The details of the “release notes” vary and we include some examples of the release notes in the Appendix. Some projects clearly declare their stable releases whereas some do not. The stability of the identified package is further confirmed by going through the release notes and news of the next release which if not a bug fixing release strengthens our belief that the identified package is stable.

**Development Time.** Once the stable release is identified, the development time is length of time between the inception date of the project and the release date of the identified stable release. For projects which have not released a stable version, the length of time between the inception date of the project and the data collection date (July 31, 2006) is recorded.



**Figure 1.** Development time of the Stable Release of OSS Projects (Number of projects = 205)<sup>1</sup>

**Code Contribution.** We use Gini index to measure the inequality of code contributions from developers in a project. Gini index is a measure of disparity or inequality of a distribution. It is a number between 0 and 1. Gini index has been used extensively to measure income and wealth inequality [15]. It has also been applied to measure participation inequality in communication among OSS developers for K Development Environment project [12]. Gini index ( $G$ ) is calculated as:

<sup>1</sup> Figure 1 includes all the projects in the sample.

$$G = \frac{1}{n-1} \left( n+1 - 2 \frac{\sum_{i=1}^n (n+1-i) y_i}{\sum_{i=1}^n y_i} \right),$$

where  $n$  is the number of distinct participators in the sample,  $y_i$  is the amount of participation associated with participant  $i$ , and  $y_i$ 's are indexed in a non-decreasing order such that  $y_i \leq y_{i+1}$ .  $GDEV$  is the Gini index for code contribution and  $y_i$  is the number of CVS commits by developer  $i$ . Here,  $GDEV = 0$  would be equivalent to a fully participative *bazaar* model with equal code contribution, and higher values of  $GDEV$  indicate larger inequalities in code contribution. Smaller values of  $GDEV$  indicate a diffused core group whereas higher values of  $GDEV$  indicate a clearly identifiable core group in code development. Gini index is scale, as well as population, independent and, hence, easily comparable across projects.

**Communication Participation.** We also use Gini index to capture the pattern of communication participation ( $GCOMM$ ) among the OSS development community.  $GCOMM$  is calculated similarly using the Gini index formula discussed earlier where  $y_i$  would represent the number of emails sent by developer  $i$ . Inequality in communication participation increases the  $GCOMM$  value. Smaller values of  $GCOMM$  indicate a more equally participative community compared to larger values of  $GCOMM$  which represent communication dominated by a selected few. Smaller value of  $GCOMM$ , than that of  $GDEV$ , is indicative of a highly participative peripheral developer group in discussion for a project.

**Communication Interactivity.** Prior studies in interactions in Usenet and mailing lists have suggested that discussion threads are indicators of the level of interactivity [12], [24]. We use two covariates to measure communication interactivity: the number of threads ( $NTHREAD$ ) and average thread depth ( $THDEPTH$ ). Both covariates represent the interactivity level of

discussions in an OSS project. We expect the communication interactivity to have a positive impact on project progress.

**Release Strategy.** OSS proponents have argued that the release early and often is a useful strategy for OSS developers to get help from the large user base. An OSS project may release packages with minor functionality additions quickly for users to test and report back the bugs. The reasoning behind this strategy is that the sooner the bugs are identified the sooner they will be fixed and potential instabilities caused by their interactions with new code can be avoided. The key to field testing of a software product is to try it out in as many different settings as possible. Raymond [18] suggests that OSS developers can leverage the law of large the number to identify and fix the bugs. Given enough eyeballs, all bugs are shallow [18], [23]. A huge user base for the software implies that – software will be tested in numerous different environments, more bugs will surface, they will be characterized and communicated efficiently to more bug fixers, the fix being obvious to someone, and the fix will be communicated effectively back and integrated into the core of the product [23]. We capture this strategy through two covariates: the number of beta releases (*NRELEASE*) and user base (*NDLOAD*). The covariate *NRELEASE* is the number of beta releases for a project prior to the stable release. The covariate *NDLOAD* is the number of cumulative downloads for beta packages prior to the stable release of the package.

**The Number of Developers.** We use the number of developers (*NDVLP*) as a control variable. It is likely participation pattern may vary with the number of participating developers. We control this effect by including *NDVLP* in the model.

**Project Size.** We use project size (*SIZE*) as a control variable. One would expect the development time to be longer if the project size is larger. We control this effect by using the number of files in the source code as an indicator of the size of project.

The definitions of the covariates are also shown in Table 2. In addition, we provide descriptive statistics of these covariates in Table 3.

**Table 2.** Definition of Covariates

Covariate	Definition
<i>GDEV</i>	Gini index of developer code contribution
<i>GCOMM</i>	Gini index of developer communication contributions
<i>NTHREAD</i>	Total number of threads in the mailing list
<i>THDEPTH</i>	Average thread depth of the mailing list for a project
<i>NRLEASE</i>	Total number of beta packages released by a project prior to the stable release
<i>NDLOAD</i>	Total number of downloads prior to the stable release
<i>NDVLPR</i>	The number of unique developers that have contributed coding
<i>SIZE</i>	Total number of files of the source code in the stable release

**Table 3.** Descriptive Statistics

Covariate	Mean	St. Dev.	Min	Max
<i>GDEV</i>	0.75	0.15	0.16	0.99
<i>GCOMM</i>	0.67	0.17	0.12	0.97
<i>NTHREAD</i>	493.93	677.23	5	4217
<i>THDEPTH</i>	2.25	0.96	1	7
<i>NRELEASE</i>	12.97	15.33	0	97
<i>NDLOAD</i>	18301.51	30229.51	0	275961
<i>NDVLPR</i>	8.61	8.92	2	63
<i>SIZE</i>	908.68	1744.13	7	14341

## 4. The Model

Consider a random variable  $T$  that represents the development time of an OSS software product for its first stable release. Let  $h(t)$  denote the hazard function of  $T$ . The hazard function specifies the probability density (over time) of releasing stable product given that the team does not have a stable release up to time  $t$ . It is defined as:

$$h(t) = \lim_{\delta t \rightarrow 0} \frac{P[t \leq T \leq t + \delta t | T \geq t]}{\delta t} = \frac{f(t)}{1 - F(t)} = \frac{f(t)}{S(t)},$$

where  $f(t)$ ,  $F(t)$ , and  $S(t)$  are the probability density function (pdf), cumulative distribution function (cdf), and survivor function of  $T$ , respectively. It can be easily verified that the three specifications,  $f(t)$ ,  $F(t)$ , and  $S(t)$ , are interrelated as follows:

$$S(t) = \exp\left[-\int_0^t h(u) du\right],$$

$$f(t) = h(t) \exp\left[-\int_0^t h(u) du\right].$$

From the above equations, it is clear that there is a one-to-one relationship between the pdf of  $T$  and its hazard function. Hence, the hazard function uniquely determines the pdf of  $T$ , and we can restrict ourselves to  $h(t)$  in studying the development time for software projects. Also, the hazard function is finite and non-negative.

Let  $h(t | X, \theta)$  represent the hazard function of  $T$  conditional on a vector of covariates,  $X$ , and an unobserved heterogeneity,  $\theta$ . Consistent with prior literature [20], the hazard function,  $h(t | X, \theta)$ , can be represented as:

$$h(t | X, \theta) = h_0(t) \cdot \psi(X) \cdot \phi(\theta). \quad (1)$$

In the above equation,  $h_0(t)$  denotes the baseline hazard,  $\psi(X)$  is a function of the covariates, and  $\phi(\theta)$  is the specification for the unobserved heterogeneity.

#### 4.1 Baseline Hazard Function

We use a very general hazard rate formulation as:

$$h_0(t) = \exp(\gamma_0 + \gamma_1 t + \gamma_2 \ln t + \gamma_3 t^2), \quad (2)$$

where  $\gamma_0, \gamma_1, \gamma_2$  and  $\gamma_3$  are parameters to be estimated.  $t$  represents the duration term, i.e., the development time.

As shown in Table 4, the above general hazard function encompasses the Exponential, Weibull, Gompertz, and Erlang-2 distributions. For example, when  $\gamma_1 = 0, \gamma_2 = 0,$  and  $\gamma_3 = 0,$   $h_0(t)$  corresponds to an exponential distribution. When  $\gamma_1 = 0$  and  $\gamma_3 = 0,$  the baseline hazard function reduces to a Weibull distribution. The baseline hazard function can also accommodate a second-order power series approximation of an Erlang-2 distribution when  $\gamma_2 = 1$  and  $\gamma_3 = \gamma_1^2 / 2.$  Thus, our formulation of the baseline hazard function is a very general representation and can accommodate a variety of baseline hazard functions. By using a general baseline hazard function, we avoid potential model misspecification and ensure that the estimated model is robust.

**Table 4. Baseline Hazard Functions**

Parametric Restrictions	Baseline Hazard Functions	Corresponding Probability Distribution
None	$\exp(\gamma_0 + \gamma_1 t + \gamma_2 \ln t + \gamma_3 t^2)$	-----
$\gamma_1 = 0, \gamma_2 = 0, \gamma_3 = 0$	$\exp(\gamma_0)$	Exponential
$\gamma_1 = 0, \gamma_3 = 0$	$\exp(\gamma_0 + \gamma_2 \ln t)$	Weibull
$\gamma_2 = 0, \gamma_3 = 0$	$\exp(\gamma_0 + \gamma_1 t)$	Gompertz
$\gamma_2 = 1, \gamma_3 = \gamma_1^2 / 2$	$\exp((\gamma_0 - \gamma_1 - \gamma_1^2 / 2) + \gamma_1 t + \ln t + \gamma_1^2 t^2 / 2)$	A second order power series approximation to the hazard function of an Erlang-2 distribution

## 4.2 Function of Covariates

The next issue is the specification of the  $\psi(X)$  function that measures the effects of covariates.

Consistent with prior literature [20], we use the following function for covariates:

$$\begin{aligned}
\psi(X) &= \exp\left[\sum_{j=1}^J X_j(\tau)\beta_j\right] \\
&= \exp\left[\beta_1 GDEV + \beta_2 GCOMM + \beta_3 \ln NTHREAD + \beta_4 \ln THDEPTH \right. \\
&\quad \left. + \beta_5 \ln NRLEASE + \beta_6 \ln NDLOAD + \beta_7 \ln NDVLPR + \beta_8 \ln SIZE\right],
\end{aligned} \tag{3}$$

where  $X_j(\tau)$  is the value of the  $j^{\text{th}}$  covariate,  $j = 1, \dots, J$ , at time  $\tau$  when the first stable release occurs (or  $\tau$  is the data collection time if by then a project does not yet have a stable release); and  $\beta_j$  is the associated coefficient. The exponentiation of the right hand side ensures non-negativity of  $\psi(X)$ .

In this study, the covariates,  $X_j(\tau)$ ,  $j = 1, \dots, J$ , include covariates defined in Table 2.

Some covariates are log transformed due to their highly skewed nature.

### 4.3 Unobserved Heterogeneity

As mentioned earlier, one needs to control for possible impact of omitted variable(s) on the dependent variable. Inference in hazard models may be subject to specification error due to unobserved heterogeneity arising from the omission of (possibly unobservable) variables that affect the hazard. For instance, the inherent complexity of developing one product may differ from another and, hence, influences the development time. A common solution to this problem is to model unobserved heterogeneity as project specific random effects. We capture the unobserved heterogeneity by:

$$\phi(\theta) = \exp[c\theta], \tag{4}$$

where  $\theta$  represents project unobserved heterogeneity, and  $c$  is the associated coefficient. This specification follows Heckman and Singer [9]. We assume  $\theta$  to have a distribution  $G(\theta)$  across projects. Heckman and Singer [9] have shown that the parameter estimates may be sensitive to the assumptions made about the distribution of  $\theta$ . Hence, we try 3 different models for the

heterogeneity distribution which are explained in next section. Substituting (2), (3), and (4) into (1), we get the following hazard rate model:

$$h(t | X, \theta) = \exp \left[ \gamma_0 + \gamma_1 t + \gamma_2 \ln t + \gamma_3 t^2 + \sum_{j=1}^J X_j(\tau) \beta_j + c\theta \right]. \quad (5)$$

#### 4.4 Likelihood Formulation and Estimation Procedure

We use standard maximum likelihood procedure to estimate the parameters of the models. The likelihood function from the above hazard function can be written as:

$$L_i(\rho | \theta) = [f(t | \theta)]^{\delta_i} [S(t | \theta)]^{1-\delta_i},$$

where  $\rho = \{\gamma_0, \gamma_1, \gamma_2, \gamma_3, \beta_1, \dots, \beta_j, c\}$  is the set of parameters to be estimates and

$$\delta_i = \begin{cases} 1, & \text{if OSS project } i \text{ has a stable release by the data collection time;} \\ 0, & \text{otherwise.} \end{cases}$$

Note that the survivor function handles the right censoring problem caused by the spells that have not ended by the data collection time. We do not need to include left censoring as the selected projects were all initialized on Sourceforge. The likelihood function in terms of hazard function can be written as:

$$L_i(\rho | \theta) = \left\{ h(t) \exp \left[ -\int_0^t h(u) du \right] \right\}^{\delta_i} \left\{ \exp \left[ -\int_0^t h(u) du \right] \right\}^{1-\delta_i}.$$

The unconditional likelihood is then obtained by integrating over the distribution of  $\theta$ , explicitly,

$$L_i(\rho) = \int_{\Theta} L_i(\rho | \theta) dG(\theta);$$

$$L(\rho) = \prod_{i=1}^N \int_{\Theta} \left\{ h(t) \exp \left[ -\int_0^t h(u) du \right] \right\}^{\delta_i} \left\{ \exp \left[ -\int_0^t h(u) du \right] \right\}^{1-\delta_i} dG(\theta),$$

where  $N$  is the number of projects.

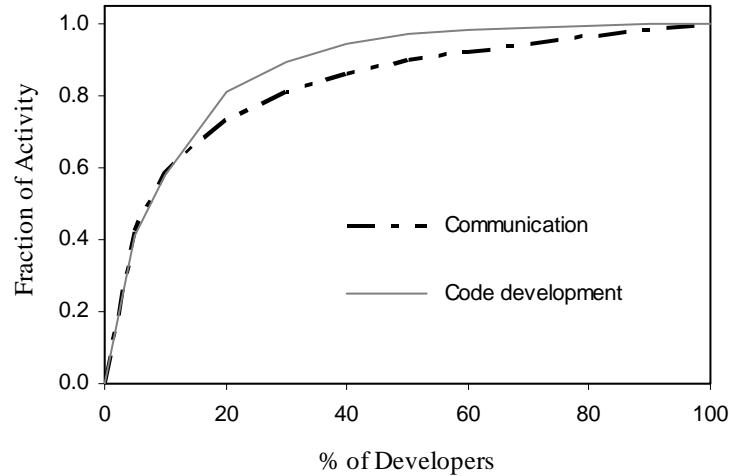
Estimation of model parameters can be carried out by directly maximizing the logarithm of this likelihood. We used sequential BFGS/Newton-Raphson algorithm to maximize the likelihood. The estimation procedure was run several times with different starting values to ensure solution stability. To mitigate the impact of misspecification of heterogeneity distribution we consider three cases: no heterogeneity (NH), standard normal heterogeneity (SNH), and non-parametric heterogeneity (NPH). In the NH model, we obtained the parameter estimates by setting  $\phi(\theta) = 1$ . In the SNH model, the parameter estimates were obtained by assuming that  $\theta$  has a standard normal distribution. This is done by numerically integrating the likelihood function over  $[-3, 3]$ . The third model involves a non-parametric estimation procedure for unobserved heterogeneity. This procedure leads to estimates that are well behaved even in modest sample sizes. This involves approximating the underlying unknown probability distribution by a finite number of support points  $(\theta_1, \theta_2, \dots, \theta_s)$ , and the location and probability mass associated with them. A common method is to set upper and lower bounds on the location of the support points. Without loss of generality, we set the bounds to be 0 and 1. We follow an iterative procedure and add support points until the inclusion of an additional point leads to a situation where two support points overlap. Note that if the SNH were truly the correct specification then the NPH should produce similar results.

## **5. The Results**

### **5.1 Distributions of Contributions**

We first examine the average contributions of the developers on both code development and communication participation for the 205 OSS projects in our sample. We find large inequalities in both code development and communication participation. As shown in Figure 2, on average,

the top 20% of the developers contributed about 81% of the source code for the sample OSS projects. This result is consistent with prior studies that a small core group of developers contribute a large percentage of coding [4], [11], [13].



**Figure 2.** Cumulative Distribution of the Developer Contributions

We can see from Figure 2 that the inequality in communication participation is slightly lower than that of code contribution. This pattern is also reflected in the values of Gini indices. As shown in Table 3, the Gini index for code contribution is 0.75, higher than the Gini index value of 0.67 for communication participation. We find, on average, the top 30% developers contributed about 81% of the messages posted on mailing lists and the most prolific 5% contributed about 42% of the messages. The bottom 50% contributed about 10% of the messages. In contrast, the bottom 50% only contributed 3% of the coding. Clearly, the participation level in communication is more dispersed compared to that in code contribution. We find that a group that is about 50% larger than the size of core development group contributed about 80% of the messages.

## 5.2 Estimated Hazard Model

The results of the parameter estimates for NH, SNH, and NPH models are presented in Table 5. Note that the coefficients of heterogeneity are significant for both the SNH and NPH models.

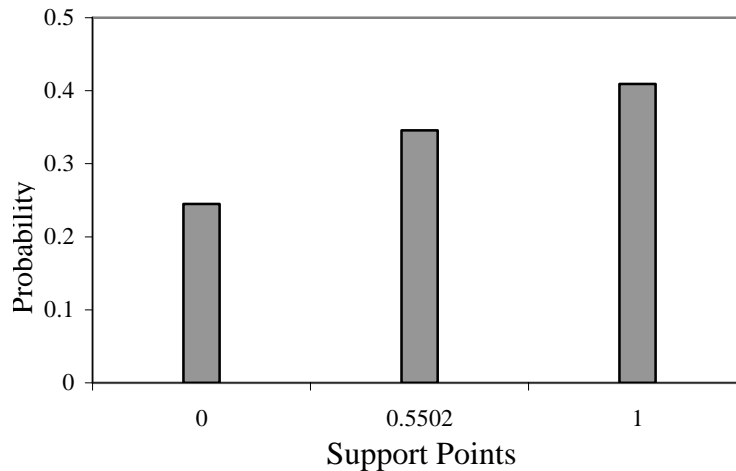
This implies that there is significant unobserved heterogeneity in the projects and the NH estimates are biased. Further, the inclusion of unobserved heterogeneity substantially increases the log-likelihood values, thereby significantly improving the model fits. The log-likelihood value is the highest for the non-parametric specification, suggesting that the NPH model outperforms the SNH and the NH model in terms of model fits.

**Table 5.** Parameter Estimates and Standard Errors (in parenthesis)

Parameters and Covariates	No Heterogeneity (NH)	Standard Normal Heterogeneity (SNH)	Non Parametric Heterogeneity (NPH)
$\gamma_0$	-4.5594 (0.7926)	-5.8508 (1.0911)	-6.9197 (0.7411)
$\gamma_1$	0.3092 (0.1354)	0.3601 (0.0943)	0.4866 (0.2030)
$\gamma_2$	-0.2796 (0.3709)	0.2376 (0.0982)	0.4017 (0.1348)
$\gamma_3$	-0.0118 (0.0048)	-0.0126 (0.0039)	-0.0143 (0.0065)
$\beta_1$ ( <i>GDEV</i> )	2.9516 (0.6600)	5.0450 (0.8198)	6.7814 (1.0665)
$\beta_2$ ( <i>GCOMM</i> )	-1.0445 (0.5799)	-1.9412 (0.5549)	-3.5620 (0.8085)
$\beta_3$ ( <i>NTHREAD</i> )	-0.0780 (0.0825)	0.1173 (0.1171)	0.1954 (0.0646)
$\beta_4$ ( <i>THDEPTH</i> )	0.4835 (0.3131)	0.7374 (0.4595)	1.2360 (0.4038)
$\beta_5$ ( <i>NRLEASE</i> )	0.4061 (0.1169)	0.5173 (0.1610)	0.4591 (0.1461)
$\beta_6$ ( <i>NDLOAD</i> )	-0.0120 (0.0465)	0.0194 (0.0601)	0.0121 (0.0171)
$\beta_7$ ( <i>NDVLPR</i> )	-0.1622 (0.1599)	-0.3939 (0.1352)	-0.7069 (0.2580)
$\beta_8$ ( <i>SIZE</i> )	-0.1845 (0.0716)	-0.3116 (0.0911)	-0.3415 (0.1320)
Heterogeneity	-----	1.2662 (0.3515)	5.1335 (0.8472)
Log likelihood	-518.22	-516.27	-502.23

Note: The following covariates were log transformed: *NTHREAD*, *THDEPTH*, *NRLEASE*, *NDLOAD*, *NDVLPR*, and *SIZE*.

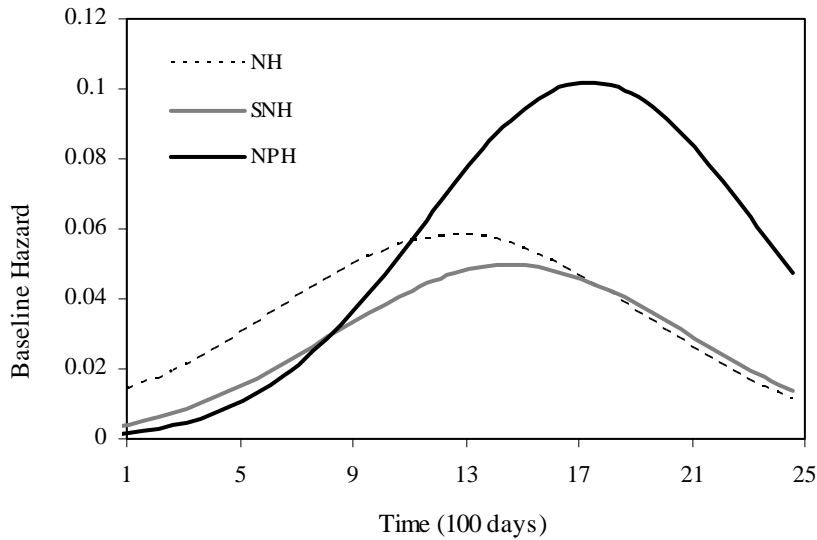
In estimating unobserved heterogeneity, the estimation procedure found three support points  $(0, 0.5502, 1)$ , to be sufficient to approximate the underlying distribution. The support points with their associated probability masses are shown in Figure 3. We can see from Figure 3 that this distribution of unobserved heterogeneity differs significantly from the standard normal distribution. This implies that the standard normal heterogeneity assumption would provide a biased estimate of the parameters in the model. Using the non-parametric approach helps uncover the underlying distribution of the unobserved heterogeneity for OSS projects.



**Figure 3.** Distribution of Unobserved Heterogeneity

We plot the baseline hazard functions for the three heterogeneity specifications. Note that the NPH baseline hazard function shows that the probability of the stable release is quite low right after the inception of the project; but it increases steeply later on as it takes time for the project to progress. We can see that the probability of a stable release increases in the first four and half years. However, the release probability drops down significantly after that period. This indicates that the OSS team is less likely to release a stable product if it has not done so in the first four and half years. This result suggests that there is a prime window for the progress of an OSS project. Within the window, the probability of success increases. However, if there are no

significant progresses within the first few years, the team could lose the momentum and the chance of a stable release declines quickly.



**Figure 4.** Estimated Baseline Hazard Functions

Compared to the NPH model, the SNH and NH hazard functions are more flat and assign a high probability for stable release at the inception of the project. However, the overall trends in the three models are similar.

### 5.3 Impacts of Covariates

We can see from Table 5 that the estimated coefficients for the covariates are sensitive to different model specifications. As discussed earlier, the NPH model outperforms both the SNH and NH models. Thus, our analysis on the impacts of covariates focuses on the NPH model.

#### 5.3.1 Contribution Inequalities

Our results show that coefficient for *GDEV* is positive and significant. High *GDEV* value indicates that a project has a core group of developers that contribute a high percentage of coding, while a low *GDEV* value suggests that developers participate in code contribution more equally (i.e. diffused core). Our results suggest that projects with a clearly identifiable core group are more efficient. This result is consistent with the findings of prior studies that successful OSS

projects usually have a core group of developers who create approximately 80% or more of the new functionality [2], [4], [13]. A project with more equal code contributions from a large pool of developers requires great effort in coordination in order to achieve the functional stability of the interrelated modules. Since developers may use their preferred coding styles, it could be difficult for others to understand or modify the code which would hamper project progress. Therefore, it may be better off to let the most “proficient” developers to code. Our results show that this contribution inequality actually leads to better project progress.

As shown in Table 5, the coefficient for *GCOMM* is negative and statistically significant. This suggests that more equal participation levels in communication can lead to shorter development time. As documented in prior research, successful OSS projects require a much larger developer group than the core to fix bugs and an even large group to report problems [4], [13]. Bugs and problems or discussions related to code development are carried out on the developer mailing lists. Besides the core developer group, a project needs a highly participative peripheral group in communication as it is the main source of bug reports, fixes, feature requests, and patch management. In addition, communication among developers is the primary source of learning, knowledge sharing, critiquing, and product redesign. More equal participation among developers means that developers are interacting on a large scale, which could lead to high level of knowledge creation and exchange in the project team even though they do not contribute to the coding directly. Overall, the level of equal participation may indicate the health of the project team and more feedbacks on the project, which will benefit the project. Thus, it is natural that more equal participation in communication facilitates project progress.

Our results suggest that the ways code contribution and communication participation affect project progress could be quite different. A core group of developers carrying out most of the

coding could be the outcome of the evolution of self-organization in OSS projects, which achieves a level of efficiency. A more equally participative community in code writing can be counter productive. However, if communications and interactions in an OSS community are dominated by a small group of people, this could lead to unhealthy group dynamics, less innovation, and lower level of knowledge sharing and feedbacks, which can hurt the project progress.

Our results imply that successful OSS projects need different types of participations. Prior studies in OSS community suggest that people participate in OSS projects for different reasons. Some are strongly motivated by the conviction that source code should be open, and some look for opportunities to improve their skills. The skill levels also vary significantly across developers. It is natural that there is a core group of developers for a project and there are people who may do less development work but still actively participate in discussions and problems reporting. Our results suggest that successful OSS projects have to not only involve a handful of core developers to get the majority of coding done, but also embrace all developers and encourage them to participate in creative thinking, discussions, and providing feedbacks.

### *5.3.2 Communication Interactivity*

We find that the number of discussion threads (*NTHREAD*) and thread depth (*THDEPTH*) both positively affect the project success. While the number of discussion threads is an indication of discussion breadth, thread depth is a measure of diverse views and the discussion depth for the topics. As the number of threads and thread depth increase, the OSS project community is engaging in healthy and highly interactive discussions. These exchanges are critical for developers to discuss a wide range of issues related to the tasks of the projects, such as system design, feature implementation, user experience, and bug fixing, which help to improve project

quality and speed of the development and testing process. In addition, threaded discussion can serve as a process of connecting new members to existing members. In order for newly joined developers to increase the chance that their posting will be contributing, they have to go through a learning process by reading the previous postings. Thus, the interactions in OSS projects enhance knowledge sharing and coordination, which are critical to the success of the projects.

### *5.3.3 Release Strategy and User Base*

The coefficient of *NRLEASE* is positive and significant which implies that the “release early and release often” strategy has a positive impact on project progress. The number of early releases can be an indication of the progress of the project. As an OSS group frequently releases its products, it signals that project is growing well. Also, frequent releases can generate valuable feedbacks that in turn help the project team to develop the stable product.

The number of downloads (*NDLOAD*) is an indication of the size of the user base and initial acceptance of the product. Although the sign of the coefficient for the number of downloads is positive, the relationship is not statistically significant. Therefore, we cannot claim that the number of downloads significantly affect the progress of the project.

### *5.3.4 Project and Team Characteristics*

For the control variables, we find project size (*SIZE*) negatively affects the likelihood that a project team releases a stable product. It simply means that the larger the project, the longer will be the development time. We also find that the number of developers (*NDVLPR*) negatively affects project progress. Since we control project size in our model, we can interpret the results as that given everything is the same, a large development team does not necessarily benefit the project progress. This may be due to the fact that coordination cost for OSS project is quite large with a large number of developers. Another explanation is that the sheer number of developers is

not crucial for project progress; what is more important is the quality of the core developers. This explanation is line with the views of Cox [2] that a successful OSS project requires only a very small number of real programmers with the time and the right kind of mental state to contribute to a project.

### 5.3.5 Critical Success Factors

A more intuitive understanding of the estimated coefficients is to examine their respective elasticity. Elasticity measures the percentage change of the hazard rate induced by one percentage change of a covariate. For a log transformed covariate, its estimated coefficient is simply the elasticity measure [22], for example,  $\partial \ln h = \beta_3 \cdot \partial \ln NTHREAD$ . We can interpret that a 1% change in the number of threads will change the rate of project stable release by about 0.2%. A 1% change of the average thread depth increases the stable release rate by about 1.2%. For these covariates, their elasticity is scale free and hence can be compared directly with each other to identify the most influential ones. However, for the covariates *GDEV* and *GCOMM* that are not log transformed, their elasticity is different from their corresponding coefficient. Since  $\partial \ln h = \beta_1 \partial GDEV = (\beta_1 GDEV) \cdot \partial \ln GDEV$ , the elasticity for *GDEV* is  $\beta_1 GDEV$ . Similarly, the elasticity for *GCOMM* is  $\beta_2 GCOMM$ . At their respective mean levels, the estimated elasticity for *GDEV*, or *GCOMM*, is 5.09, or 2.39.

Comparing the estimated elasticity measures, we find the following factors are associated with OSS projects that do not progress well: (i) lack of a core developer group, (ii) dormant peripheral group in communication, (iii) low communication interactivity exemplified by the lack of depth in threaded communications, and (iv) over-dependence on internal community for beta testing. This observation is confirmed by a careful re-look at *GDEV*, *GCOMM*, *THDEPTH*, and *NRLEASE* for the projects that have not released stable version by the data collection time.

For those projects, the average *GDEV* (0.54), *THDEPTH* (1.1), and *NRLEASE* (6.71) are lower than the corresponding overall averages (0.75, 2.25, and 12.97, respectively) shown in Table 3, whereas the average *GCOMM* (0.79) is higher than the overall average *GCOMM* (0.67).

## 6. Conclusions

This study uses project data from Sourceforge to empirically examine how the OSS development characteristics affect project progress. Prior case studies have examined the successful OSS projects and found that OSS projects need to have a core group of developers and a much larger group for bug fixing and problem reporting [4], [13]. This study extends prior studies in this area. We collect a large sample of 205 OSS projects and use hazard models to examine the effects of code contribution, communication participation, and release strategies on project progress.

Consistent with prior studies, we find, on average, the top 20% of the developers contributed about 81% of the source code for the sample OSS projects. The top 30% developers contributed about 81% of the messages posted on mailing lists and the most prolific 5% contributed about 42% of the messages. Overall, the participation level in communication is more equal compared to that in code contribution.

Our results suggest that projects with a core group of developers conducting most of the coding are more efficient in project progress. However, OSS projects with a small group dominating communications and discussions often do not progress well. Our results suggest that the ways code contribution and communication participation affecting project progress are different. While a core group of developers carrying out most of the coding may be an efficient way to organize code development, over-concentration in communications and discussions could hamper healthy exchanges, knowledge sharing, and feedbacks, and thus negatively affect project

progress. Our results imply that while it is critical to have a handful of core developers responsible for code development, OSS projects should also embrace all developers and encourage them to participate in creative thinking, discussions, and providing feedbacks.

In addition, we find high level of communication interactivity, as measured by the number of email threads and average thread depth, significantly affects project progress. The number of beta releases is a good predictor of stable release of the product. We also find that the total number of developers and the size of the project are negatively linked with project progress. We did not find the number of early downloads to significantly affect project progress.

We find the following factors that are linked to OSS projects that do not progress well: (i) lack of a core developer group, (ii) large uneven communication participation from all project developers, and (iii) low communication interactivity exemplified by the lack of depth in threaded communications, and (iv) over-dependence on internal community for beta testing.

Future research could examine other measures on OSS success, including viability of OSS projects and software quality such as defect rates and user satisfaction. Research could also examine whether there are optimal levels of developer participation in both code development and communications.

## **References**

- [1] R. Agarwal and B.L. Bayus, "The Market Evolution and Sales Takeoff of Product Innovations," *Management Science*, vol 48, no. 4, pp. 1024–1041, 2002.
- [2] A. Cox, "Cathedrals, Bazaars and the Town Council," *Slashdot.Org*, 1998.
- [3] K. Crowston, H. Annabi, and J. Howison, "Defining Open Source Software Project Success," *Twenty-Fourth International Conference on Information Systems*, 2003.

- [4] T. Dinh-Trong and J. Bieman, “The FreeBSD Project: A Replication Case Study of Open Source Development,” *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, 2005.
- [5] R. Fielding, “Shared Leadership in the Apache project,” *Communications of the ACM*, vol. 42, no. 4, pp. 42–43, 1999.
- [6] L. Gasser, W. Scacchi, G. Ripoché, and B. Penne, “Understanding Continuous Design in F/OSS Projects,” *16th International Conference on Software Engineering & its Applications (ICSSEA-03)*, Paris, France, 2003.
- [7] M. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study,” *Proc. International Conf. Software Maintenance*, pp. 131–142, 2000.
- [8] M. Gort and S. Klepper, “Time Paths in the Diffusion of Product Innovations,” *The Economic Journal*, vol. 92, pp. 630–653, 1982.
- [9] J. Heckman and B. Singer, “A Method for Minimizing the Impact of Distributional Assumptions in Econometric Models for Duration Data,” *Econometrica*, vol. 52, pp. 271–320, 1984.
- [10] P. Hougaard, “Survival Models for Heterogeneous Population Derived from Stable Distributions,” *Biometrika* vol. 73, no. 2, pp. 387–396, 1986.
- [11] S. Koch and G. Schneider, “Effort, Cooperation, and Coordination in an Open Source Software Project: GNOME,” *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.
- [12] G. Kuk, “Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List,” *Management Science*, vol. 52, no. 7, pp. 1031–1042, 2006.

- [13] A. Mockus, R. Fielding, and J. Herbsleb, “Two Case Studies of Open Source Software Development: Apache and Mozilla,” *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.
- [14] B. Nonneke and J. Preece, “Lurker Demographics: Counting the Silent,” *Proc. SIGCHI Conf. Human Factor Computer Systems*, ACM Press, New York, pp. 73-80, 2000.
- [15] M. Paglin, “The Measurement and Trend of Inequality: A Basic Revision,” *American Economic Review*, vol. 65, no. 4, pp. 265–266, 1975.
- [16] J. Paulson, G. Succi, and A. Eberlein, “An Empirical Study of Open-Source and Closed-Source Software Products,” *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 246-256, 2004.
- [17] B. Rajagopalan and B. Bayus, “Exploring the Open Source Software Bazaar,” University of North Carolina Working Paper, 2004.
- [18] E. Raymond, *The Cathedral and the Bazaar*, O’Reilly, Sebastopol, CA, 1998.
- [19] P. Singh, N. Youn, and Y. Tan, “Developer Learning Dynamics in Open Source Software Projects: A Hidden Markov Model Approach,” University of Washington Working Paper, 2006.
- [20] G. Van Den Berg, “Duration Models: Specification, Identification, and Multiple Durations,” in J.J. Heckman & E.E. Leamer (ed.) *Handbook of Econometrics*, Chapter 55, pp. 3381–3460, 2001.
- [21] P. Wayne, *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans*, New York: Harper Collins, 2000.
- [22] J. Wooldridge, *Econometric Analysis of Cross Section and Panel Data*, The MIT Press, 2002.

- [23] S. Weber, *The Success of Open Source*, Harvard University Press, 2004.
- [24] S. Whittaker, L. Terveen, W. Hill, and L. Cherny, “The Dynamics of Mass Interaction,” *ACM 1998 Conf. Computer Supported Cooperative Work*, ACM Press, New York, pp. 257-264, 1998.
- [25] Y. Yamauchi, M. Yokozawa, T. Shinohara, T., and T. Ishida, “Collaboration with Lean Media: How Open Source Software Succeeds,” *ACM 2000 Conf. Computer Supported Cooperative Work*, ACM Press, New York, pp. 329-338, 2000.

## Appendix. Release Notes Examples

### Example 1

**Release Name:** 2.0.0

**Notes:**

This is the first stable release of GnomeSword2 for the GNOME2 platform. It requires GNOME 2.4 and version 1.5.6 or 1.5.7 of the SWORD libraries.

This version of GnomeSword represents a major rewrite and features a full port to GTK2, with a new GUI. It uses more GNOME functionality, and has support for new SWORD features such as preverse headings. A manual is included, in both the English and French languages.

There have been huge usability improvements and numerous other enhancements since the previous GNOME 1.x version (the stable 0.7.x series, which is now deprecated). We encourage you to take a look at the new version, which is now our stable platform on which all new development will be taking place.

**Changes:**

2003-12-24 Andy <andyp@users.sf.net>

- bumped to version 2.0.0 (yay! a real stable release!)
- added French manual (thanks to Dominique Corbex)
- updated French translation
- disabled out-of-date translations for GNOME2 build
- (only en\_GB and fr\_FR are current)
- cleaned English manual
- added screenshots for English manual

2003-12-23 Terry <tbiggs@users.sf.net>

- added new sidebar menu button
- cleand up the html.c code
- changed <back> and <foward> buttons on nav toolbar to stock buttons

2003-12-21 Terry <tbiggs@users.sf.net>

- fixed bug #863854 that I caused when I fixed to build with -  
DGTK\_DISABLE\_DEPRECATED -- still builds with -DGTK\_DISABLE\_DEPRECATED

----  
----  
----

2003-07-25 Terry <tbiggs@users.sf.net>

- added src/gnome2 for port to gnome2

## Example 2:

**Release Name:** 0.9.1 DEV ONLY! NOT FOR PRODUCTION USE!

**Notes:**

Many huge nasty bugs fixed, many minor bugs remain

## Example 3

**Release Name:** 2001-11-02

**Notes:**

First octave-forge release, rev c. Mostly administrative changes, with a few bug fixes thrown in. See octave-forge/ChangeLog for all the details.

**Changes:**

mex : fix zero termination bug  
fsolve : allow arguments to be passed through to the user function  
trisolve : slow m-file version for those who can't use trisolve.oct  
fminbnd : allow arguments to be passed through to the user function  
nchoosek : extend valid input range  
xcorr,xcov : fix argument processing  
lin2mu,mu2lin : restore default Octave behaviour