The Journal of Systems and Software 85 (2012) 895–905

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Factors affecting the success of Open Source Software

Vishal Midha [a,*], Prashant Palvia [b,1]

[a] The University of Texas – Pan American, 1201 W. University Drive, Edinburg, TX 78539, United States
[b] The University of North Carolina at Greensboro, 1000 Spring Garden St, Greensboro, NC 27402, United States

ARTICLE INFO

Article history:
Received 20 July 2010
Received in revised form 3 November 2011
Accepted 8 November 2011
Available online 17 November 2011

Keywords:
Open Source Software
System success
Longitudinal effects
Intrinsic cues
Extrinsic cues

ABSTRACT

With the rapid rise in the use of Open Source Software (OSS) in all types of applications, it is important to know which factors can lead to OSS success. OSS projects evolve and transform over time; therefore success must be examined longitudinally over a period of time. In this research, we examine two measures of project success: project popularity and developer activity, of 283 OSS projects over a span of 3 years, in order to observe changes over time. A comprehensive research model of OSS success is developed which includes both extrinsic and intrinsic attributes. Results show that while many of the hypothesized relationships are supported, there were marked differences in some of the relationships at different points in time lending support to the notion that different factors need to be emphasized as the OSS project unfolds over time.

1. Introduction

Over the last two decades, the practice of community based software development and distribution, combined with novel uses of intellectual property law, has emerged. This practice has come to be known as "Open Source Software" (OSS) development. The success of community based model of software development has extended this paradigm into new arenas such as pharmaceutical development, knowledge repositories, space exploration, and education as well as to unexpected domains such as developing recipes for open-cola, coffee and beer (Lawton, 2002). The hype of open source phenomenon has also led to an exponential growth in the number of such projects. However, except for a few, the majority of the projects at open source hosting sites have been unsuccessful (Chengalur-Smith and Sidorova, 2003). This apparent paradox has generated immense interest among open source researchers in identifying the factors that affect open source project success.

A typical open source project starts when an individual (or group) feels the need for a new feature or entirely new software, and someone in that group, eventually writes one. In order to share it with others who have similar needs, the software is released under a license that allows the community not only to use it but also to see the source code and modify it to meet local needs and improve the product by fixing bugs (Crowston et al., 2006; Feller and Fitzgerald, 2002; Midha et al., 2010). Making software available widely on an open network, e.g., the Internet, allows developers around the world to contribute code, add new features, improve the present code, report bugs, and submit fixes to the current version. The developers of the project incorporate the features and fixes into the main source code and a new version of the software is made available to the public. This process of improvement and customization through code contribution and bug fixing is continued in an iterative manner (Midha et al., 2010). Clearly, as OSS projects are continually in development, the project characteristics change through the life of the project as different versions are released. In addition, even the interests of its user and developer base can change. It is, therefore, important to understand the changing dynamics of the relationships between project success measures and their antecedents over time.

The purpose of this research is to understand the impact of various factors, categorized as intrinsic and extrinsic factors, on OSS project success over the first three years of its life. A longitudinal analysis of these factors is conducted at various stages in the OSS life cycle. This analysis provides unique insights into various project management decisions, such as the escalation of errors and design flaws leading to substantial maintenance costs. As a result, OSS project administrators can develop a deeper understanding of the metrics for open source development that can help them to manage the escalation of costly flaws in these projects. Ultimately, the understanding of the antecedents could lead to a project's success both in terms of market penetration and technical achievements over time.

It may be noted that some of the relationships in this research, such as impact of complexity on technical success (Feller and Fitzgerald, 2002), and impact of license type on market success

* Corresponding author. Tel.: +1 956 381 2801; fax: +1 956 381 3367.
E-mail addresses: vmidha@utpa.edu (V. Midha), pcpalvia@uncg.edu (P. Palvia).
[1] Tel.: +1 336 334 4818; fax: +1 336 334 4550.

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.11.010

(Stewart et al., 2005), have already been investigated in the OSS literature. They are, however, still considered in this study for two reasons: (i) completeness of the model, and (ii) to study the impact of these variables in a longitudinal fashion, which to the best of our knowledge has received very limited attention. One example is that of Subramaniam et al. (2009) who modeled time-variant and time-invariant variables to study their impact over time. Their study could be considered the first quantitative longitudinal study to investigate factors that impact the success of OSS projects. Our study builds on Subramaniam et al's (2009) work to propose a comprehensive model by including various other factors, such as complexity, modularity, responsibility assignment, and language translations.

The remainder of the article is organized as follows: the next section presents theoretical foundations leading to the hypotheses and model development. It is followed by a description of methods and measures. The following sections present the evaluation of the model and discussion of the results. The article concludes by acknowledging its limitations and highlighting its contributions to both research and practice.

2. Theory and research hypotheses

2.1. OSS success

The meaning of success in the software development context is subjective. Some researchers define success of OSS projects in terms of the size of consumer base, whereas others have defined it as the number of free contributions from its developer community (Feller and Fitzgerald, 2002; Stewart et al., 2005; Crowston et al., 2006). For example, Raymond (2001) points that OSS projects success is characterized by a continuing process of volunteer developers fixing bugs, adding features and releasing software "often and early". Supporting Raymond's definition, Markus et al. (2000) suggested that it is critical to attract contributors on an on-going basis to keep the project sustainable. English and Schweik (2007) grouped projects in successful and failures based on initiation and growth in terms of number of releases and downloads. Feller and Fitzgerald (2002) observed that, for well known open source projects such as Linux and Apache, market penetration could be used as a success measure. However, this measure is not applicable to less known projects. For lesser known projects, success is indicated by the popularity of the project among its current and potential users (Stewart et al., 2005).

In congruence with Crowston et al. (2006) suggestion that success is a multi-dimensional construct that needs to be assessed from multiple perspectives, Grewal et al. (2006) pointed out that measuring success of OSS projects in terms of technical achievements or market success represents an incomplete picture of success. They further pointed that a comprehensive picture of OSS success should encompass both the developers' technical achievements as well as indicators of market success. This pair of criteria for project success is consistent with the literature on software success (Rai et al., 2002) and new product development (Mansfield and Wagner, 1975). Following these recommendations, we focus our attention on both commercial and technical measures of success. '*Market Success*' of an OSS project, a measure of project popularity, is defined as the level of interest displayed in the project by its consumers, whereas '*Technical Success*' is defined in terms of developer activity, i.e., the level of effort expended by developers of the project.

As there are over 300,000 OSS projects listed on SourceForge alone (October 2011), then the question arises: which projects achieve market and technical success? What factors govern the two success criteria? These need to be evaluated from the consumer's point of view. There are two types of consumers: users and developers (users simply use OSS products, whereas developers use and contribute to the development of OSS products). More specifically, (i) which projects users display interest in, (ii) which projects OSS developers contribute to, and (iii) as most OSS projects are continually in development, implying that the project characteristics change over time, then how does this impact OSS users affinity and OSS developers activity in the project. As OSS users and developers have different motivations to be associated with OSS projects, we argue that two different sets of factors exist that impact the two consumer types. We borrow the concepts of intrinsic and extrinsic cues from Cue Utilization Theory (CUT) to explain the two sets of factors, and how they impact the success of OSS projects.

2.2. Cue utilization theory and Open Source Software

As per cue utilization theory (CUT), products are conceived to consist of several cues. Each cue provides a basis for developing various impressions of the product, and is used by consumers in their product selection decision (Olson, 1972). Cues can be classified as extrinsic or intrinsic. *Extrinsic cues* are product-related attributes which are not part of the physical product. Conversely, *intrinsic cues* represent product-related attributes which cannot be manipulated without also altering physical properties of the product. Intrinsic cues play a major role in the product selection decision-making process. Nevertheless, consumers depend more on extrinsic cues as "surrogates" for risk reduction, as intrinsic cues require more time and effort (Zeithaml, 1988).

Careful evaluation of SourceForge, the biggest OSS repository, and the OSS literature reveal a select yet diverse set of cues which are available to consumers. Following Olson's (1972) definition, the set of cues was grouped into six extrinsic and two intrinsic cues. The product-related attributes which are not part of the physical product, such as license type, available language translations, size of existing user, size of developer base, and responsibility assignment of work were categorized as extrinsic cues; whereas product-related attributes which represent the properties of the product, such as complexity and modularity of the source code were categorized as intrinsic cues.

According to CUT, consumers arrive at judgments by evaluating intrinsic and extrinsic cues. However, Wright (1975) pointed out that consumers tend to simplify their decision making process by basing their judgments on summary information available through external cues, rather than on the total set of project's attributes, including both internal and external cues. This is most likely for choice behavior where consumers prefer to engage in minimal processing. Although, in the case of OSS projects, the complete source code is available for inspection to all consumers, the OSS users may not possess the necessary skills and knowledge to evaluate the intrinsic cues accessible through the source code. Even if they have the necessary skills, they may rely on more easily interpretable extrinsic attributes than spending time and effort on understanding and evaluating the source-code. Consequently, extrinsic cues may form the basis of selection of OSS projects by its consumers.

Conversely, knowledgeable consumers such as OSS developers are less likely to use extrinsic cues alone in their decision making process (McConnell, 1968). Due to their well-developed cognitive structures, experts prefer to use both extrinsic and intrinsic cues to form their decision (Olson, 1972). The OSS developers not only use OSS products, but also contribute to their source code development. Such consumers have the necessary skills and knowledge to evaluate the intrinsic cues of OSS projects. Therefore, both intrinsic and extrinsic cues form the basis of project's technical success. Combining these arguments with H1 (as discussed in the next section) leads to the development of our comprehensive theoretical OSS success model (Fig. 1).

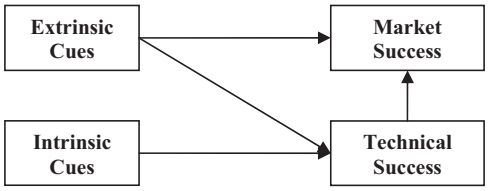

Fig. 1. Theoretical OSS success model.

3. Hypotheses development

3.1. Extrinsic cues

When a potential consumer visits a software project's homepage on SourceForge, the consumer can locate various cues that might help him or her in selecting projects. These cues may be unrelated to the actual functionality or working of a project, and the consumer may not require special technical skills or knowledge to interpret these cues. In the following section, we explain how these extrinsic cues are utilized by the OSS consumers in their selection decision.

3.1.1. Technical success and market success

A project will have high technical success, if developers devote time to making contributions and developing enhanced software versions. Whenever any bug is fixed or a feature is added or a milestone is reached by a project, the changes are *generally* revealed to the OSS community in the form of announcements. Source Forge and FreshMeat, list project's announcements on their home page. Therefore, any project that makes frequent announcements gets listed frequently on the repository's home page. Such frequent listings on the home page act as that project's advertisements to anyone visiting the webpage. These advertisements are extrinsic cues that require minimal processing by consumers, and have been shown to have a mental effect (e.g., awareness, memory, favorable attitude towards the product) on consumers (Vakratsas and Ambler, 1999).[2] Thus, these projects have the potential of becoming successful in the market, resulting in our first hypothesis:

H1. Technical success is positively related to market success.

3.1.2. License choice

The Open Source Initiative (OSI) lists over a few dozen types of licenses that comply with Open Source Development. These licenses have been grouped into different categories on the basis of restrictiveness (Lerner and Tirole, 2002, 2005; Scacchi, 2004). Following Lerner and Tirole (2005), we use the term 'restrictive' to refer to licenses that have both the copyleft[3] and viral[4] provisions, and 'non-restrictive' to refer to licenses that do not have both provisions.

While the copyleft and viral provisions act to maintain the openness of the code, they are restrictive in the sense that they limit what a user can do with the software (Stewart et al., 2005). These provisions may restrict commercialization of OSS applications (Lerner and Tirole, 2002; West, 2003) and reduce the perceived usefulness of the software for potential users that seek to advance commercial interests (Stewart et al., 2005). Additionally, such restrictions limit the use of the software in conjunction with other software that is distributed under different licenses. For example, someone working on a non-restrictive license based project would not be able to use restrictive license based software projects; and that decreases consumer's interest in the restrictive projects. Hence, we hypothesize:

H2a. OSS projects that use a non-restrictive license exhibit higher market success than those that use a restrictive license.

In contrast, the commercialization of an open source project may not be perceived as beneficial by the developers of that software. If Open Source Software gets commercialized, the developers could lose visibility in the open source community. In an extreme but possible scenario, if the commercial version of the software becomes dominant, the developers may end up having to pay for the software that grew out of their own efforts and at the same time may not be able to customize the commercial version to best suit their needs (Stewart et al., 2005). Thus, at least two sources of motivation, personal utility and recognition among peers, may be muted when nonrestrictive licenses are employed, leading to a decreased interest from the developer community. Thus,

H2b. OSS projects using a non-restrictive license exhibit lower technical success than those using a restrictive license.

3.1.3. Consumer base

In the absence of objective information, such as reviews from existing user base, potential users need to try out software in order to evaluate its usefulness. The information cascades theory suggests that the existing consumer base could be among the extrinsic cues used by potential users in selecting OSS projects (Bikhchandani et al., 1998). Informational cascades refer to the situation "when it is optimal for an individual, having observed the actions of those ahead of him, to follow the behavior of the preceding individual without regard to his own information". Such a situation arises when decision makers have imperfect knowledge of the true value of a product; so they infer its utility from observing actions of their predecessors. The influence of others' behavior can be so substantial that it dominates the influence of their own knowledge. In this case, decision makers would imitate their predecessors without regard to their own information. Therefore, we propose that the size of existing consumer base of a project will impact the attractiveness of the current version of the OSS project to its consumers. von Hippel and von Krogh (2003), however, argued that users and developers have different sense of belonging. For example, active developers identify themselves into the well integrated community of active developers while the users have their own community. Thus each group will have its own effect on market success. Thus,

H3a. The cumulative existing user base of the previous versions of an OSS project is positively associated with its current version's market success.

H3b. The cumulative existing developer base of the previous versions of an OSS project is positively associated with its current version's market success.

From his observations on Linux and Fetchmail, Raymond (2001) emphasized the importance of large groups: "Given enough eyeballs, all bugs are shallow ones". This function of group size certainly has been a factor in the success of high-profile OSS products like Linux and Apache Web server. Because of large team size, bugs and errors can be rapidly spotted and fixed, leading to our next hypothesis,

H3c. The cumulative existing developer base of the previous versions of an OSS project is positively associated with its current version's technical success.

[2] SourceForge used to post the update announcements on the home page at the time of data collection, and does not post them at this time.

[3] Copyleft provision requires that modified versions of the software should also be open.

[4] Viral provision requires that source code should be combined with the source code of other software that is distributed under licenses that share the copyleft requirement.

3.1.4. Language translations

Research has shown that native language interface is an important determining factor of the effectiveness of software usage. Even a simple translation to native language will increase the productivity of users of commercial systems (Bodley, 1993). Davis et al. (1989) definition of perceived ease of use is "the degree to which a person believes that using a particular system would be free of effort", and effort may clearly be seen as a cost associated with using the software in a language different than the user's native language. The availability of a preferred language translation acts as an external cue that helps a consumer evaluate the value of a product. From these arguments, we can deduce that the availability of software in different languages attracts more users, leading to greater market success of the software. Thus,

H4. The number of language translations of an OSS project is positively related to its market success.

3.1.5. Responsibility assignment

In OSS projects, when new bugs are reported or new patches and features are requested, the tasks to fulfill the request are delegated either to particular developer(s) or not delegated to anyone at all. In the latter case, the responsibility remains open such that anyone among the entire community could take that responsibility.

OSS thrives primarily on voluntary contributions from OSS developers. This has serious implications for OSS. No one can be forced to work on tasks that do not interest them. Project leaders will assign responsibilities for specific tasks to be completed. Depending on motivation, individuals may or may not complete the task. However, as opposed to traditional chains of command in traditional software development, OSS development teams utilize developers' desire to gain and maintain a good reputation as a mechanism to make sure that the project continues on track (Markus et al., 2000; Raymond, 2001). In this reputation system, failures and irresponsible actions have consequences, such as loss of status for the developers. Therefore, when a task is assigned to a developer, the developer not only feels responsible but also fears the loss of social status in case of not completing the assigned task. Thus, we argue that when the project administrators assign the responsibility of a task to a developer, the fear of loss of status motivates the developer to complete the task without unnecessary delays. In other words, technical success should be directly proportional to responsibility assignment. Hence,

H5. OSS projects that delegate responsibility exhibit higher technical success.

3.2. Intrinsic cues

As opposed to extrinsic cues, consumers require special skills and more than minimal processing to evaluate products based on their intrinsic cues. In the following section, we explain how these intrinsic cues, such as complexity and modularity of the source code, impact the OSS consumers selection decision.

3.2.1. Complexity

This literature suggests that complexity has multiple facets, including structural complexity (Gorla and Ramakrishnan, 1997) and algorithmic complexity (Darcy et al., 2005). The structural complexity of a program comes from "the organization of program elements within a program" (Gorla and Ramakrishnan, 1997), and algorithmic complexity is defined in terms of the time taken to execute a program (Darcy et al., 2005). Dealing with structural complexity primarily expends intellectual resources; whereas algorithmic complexity primarily consumes machine resources. Due to exponential improvements in technology, the ever declining cost of per unit machine resource has rendered algorithmic complexity unimportant as compared to structural complexity.

Kearney et al. (1986) suggested that the difficulty of understanding depends, in part, on structural properties of the source code. As it is not unusual that a developer contributing to the source code has not participated in the development of the original program, a large amount of the developer's efforts goes into understanding and comprehending the existing source code (Kemerer, 1995). Comprehending existing source code, which involves identifying the logic in and between various segments of the source code and understanding their relationships, is essentially a mental pattern-recognition by the software developer and involves filtering and recognizing enormous amount of data (Rilling and Klemola, 2003). When a developer has to deal with a source code with high structural complexity, he or she has to frequently search among dispersed pieces of code to determine the flow of logic (Ramanujan and Cooper, 1994). Consequently, the time needed to comprehend source code and, ultimately, contribute to the source code increases as the level of complexity increases. Hence, we propose that the level of developers' contribution decreases as the level of structural complexity increases.

H6. Complexity of the OSS project is negatively related to its technical success.

3.2.2. Modularity

Modularity is extremely critical in OS development. As mentioned by Torvalds (1999), "for without it [modularity], you cannot have people working in parallel". With a modular design, multiple programmers can work to build new functions into the same module. It simplifies software design by splitting large complex problems, and increases flexibility by splitting the project into smaller sub-parts. The high modular nature of OSS allows development of specific parts of the system in autonomy and without need to coordinate efforts with other sub-parts. Modularity also allows development to continue while avoiding a situation where the impact of one person's enhancements to a module leads to problems with the work in some other module. The increased flexibility helps in designers' better understanding of the problem, in turn, increasing the probability of them contributing to the project. As a result, the overall level of contributions from OSS developers increases. Hence, we hypothesize:

H7. Modularity of the OSS project is positively related to its technical success.

The complete research model developed by combining the above arguments is presented in Fig. 2.

4. Methodology and data collection

As we are interested in studying success factors over a longitudinal period, we evaluated projects at different stages of their life. We selected four time periods to represent the four stages noted by Wynn (2003), and Schweik and Semenov (2003).[5] For each project, data was collected for the first version ($T = t1$) representing the initiation stage; a version released within 3 months of the first version ($T = t2$) and another version released within 6 to 9 months of the first version ($T = t3$) representing intermediate growth stage; and, yet another version within 2–3 years of the first version ($T = t4$) representing growth stage (English and Schweik,

[5] An alternate approach to study longitudinal evolution of projects would be to group projects by the number of releases (English and Schweik, 2007; Wiggins and Crowston, 2010) and study how various factors impact project success at various number of releases.

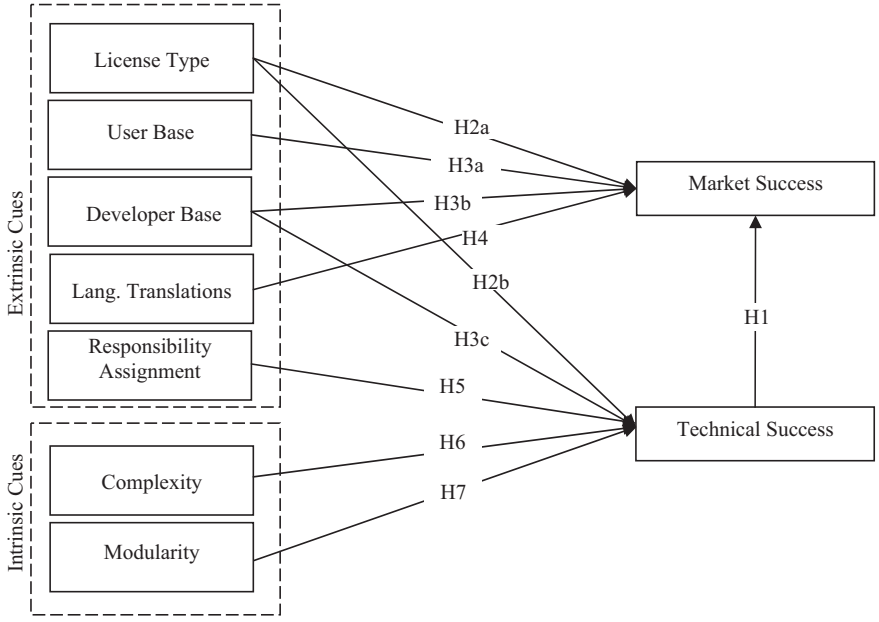

Fig. 2. Conceptual research model and hypotheses.

2007; Wiggins and Crowston, 2010). If a project had more than one version released within the above time frames, the latest version was selected. Grouping projects by evolution time allows us to study how various independent variables impact OSS success at different project ages, irrespective of the number of releases a project may have had. It may be noted that these time periods might not exactly imitate the four stages, but these allow us to empirically study the OSS success factors over a longitudinal period without introducing bias due to project age. In other words, this approach allows us to study the success factors during the first three years of development. Accordingly, a project must have released a minimum of four different versions during its first three years of life to be included in this study. Therefore, all projects that were registered with SourceForge between January 2000 and December 2004 and were active 2–3 years later were included in the study. The final data collection was completed in January 2008.

4.1. Sample

Several strategies were employed in the sampling procedure to control for possible extraneous effects of confounding variables, to increase the homogeneity of the sample, and to increase the internal validity of the findings. The sample was restricted to projects that were targeted to either end users or developers, but not both in order to avoid ambiguity. The sample was further restricted to projects written wholly or partially in the C++ programming language and those designed for the Microsoft Windows operating system.[6] This decision was based on prior findings that coders' choice of programming language influences program size and program complexity (Jones, 1986), and hence code written in different programming languages are not directly comparable. Software written in "low" level programming languages tend to have more lines of code and take longer to understand, correct, or extend than those written in "high" level languages. Likewise, programming efforts tend to vary with the operating system of the project. We chose the Windows operating system because of its widespread deployment and popularity and because of the large number of open source projects on this computing platform. Using the above criteria, 887 projects were randomly selected.[7] Attempts were made to measure values for all variables of interest for all the four versions of each project. However, not all variables could be measured owing to certain limitations, such as unavailable source files, or missing data on project's web page. As a result, 283 projects were left for which all data were available for the different versions of interest.

4.2. Operationalization

Market success: Market success is measured by using the number of downloads as a surrogate. This measure has been consistently and widely used in studying OSS project's popularity (Grewal et al., 2006; Rai et al., 2002). Market success for the *N*th version was measured by the total number of times the *N*th version was downloaded. This is directly measured through the project's statistics page on SourceForge's repository as the number of downloads.

Technical success: OSS development teams use a Concurrent Versioning Systems (CVS), such as SVN or GIT, to manage the software development process. CVS enables team members to store code at a central location. This enables members to retrieve the source code to make changes. The CVS keeps track of every change, including what was changed, *when* it was changed and *who* made the change. This helps developers work in parallel and not overwrite other developer's work accidentally. A commit occurs when a developer uploads an altered source code file. Based on Grewal et al. (2006), we treat a commit to be a successful technical refinement and a measure of technical success. For each project, technical success for the *N*th version was measured by the total number of commits made in the source code of the $N-1$th version (before the modified *N*th version is made available).

[6] With the listed criteria, we filtered 41,532 projects with Windows OS, and 23,916 projects with C++ programming language. Out of these, 14,943 projects matched the requirements of Windows OS and C++ language.

[7] All the qualifying projects numbers were listed in Excel, and, then through random number generation function, the first 887 projects were selected. The number 887 was also randomly selected through a random number generation function.

Existing consumer base: There are two types of consumers: users and developers. The size of existing user base was measured as the (cumulative) number of times the project has been downloaded until the $N-1$th version. The (cumulative) size of developer base was measured as the unique number of developers that have contributed to the development of the project until the $N-1$th version. The unique developers for each project were extracted by parsing CVS log files.

Complexity: There are a variety of complexity metrics available in literature and being practiced by the software industry. The two most common metrics are Halstead's E and McCabe's cyclomatic. We looked at complexity as the degree of cognitive effort involved. Halstead (1977) measures completely ignore the factor corresponding to complexity of function calls. They are suitable only for predicting algorithmic complexity of a program. If a program has a very small number of operators but many function-calls, then these metrics will not give correct estimation (Rana et al., 2006). In such cases, these metrics will classify an application as difficult to comprehend.

McCabe's Cyclomatic complexity is another measure of cognitive complexity. It tends to assess the difficulty faced by the maintainer in order to follow the flow control of the program. It is considered an indicator of the effort needed to understand and test the source code (Stamelos et al., 2002). Kemerer and Slaughter (1997) used the McCabe cyclomatic complexity metric to evaluate decision density, which represents the cognitive burden on a programmer in understanding the source code. Therefore, in order to measure the complexity of the task, we measured the cyclomatic complexity of the corresponding version of the project and subjected the source code files to CCCC, an OSS analysis tool. The complexity of source code of $N-1$th version was measured to study its impact on the technical success of the Nth version. To account for the effects of size, the complexity metric was normalized by dividing it by the number of lines of code for the project. This procedure also reduces collinearity problems when size is included in regression models (Gill and Kemerer, 1991).

Modularity: Parnas (1972) defines modularity as the number of modules in software. A module, in this research, is defined as a group of member functions, including classes, namespaces, and interfaces. Member functions are functions that are declared as members of a class. To measure the number of modules, again the same metric tool was used. Similar to complexity, the modularity of source code was recorded for the $N-1$th version. Capiluppi et al. (2003) have also used the same measure in OSS research.

License selection: Lerner and Tirole (2005) categorized open source licenses as highly restrictive (e.g., GPL) and relatively less restrictive. Using the same classification, licenses were divided into two categories, namely GPL project (restrictive = 1) and others (non-restrictive = 2).

Project translations: All announcements related to language translations by date for each project were recorded. Additionally, we scanned change log files and calculated the number of language translations for each version from the recorded dates.

Responsibility assignment: Each project maintains its own list of tasks that are either assigned to a specific developer or left unassigned so that anyone from the OSS community can work on it. The responsibility assignment for the Nth version was calculated as the fraction of the number of tasks assigned to someone over the total number of tasks listed for $N-1$th version after its release.

Project release: In addition to the above variables, we used project release as a control variable. We distinguish between two kinds of releases: major releases and minor releases. A major release indicates that there has been a substantial change in the software, whereas a minor release indicates that less significant changes have occurred from the previous release. Using this classification, we recorded minor release as 1 and major release as 2.

Table 1
Summary statistics for all time periods.

Statistic	T = t1	T = t2	T = t3	T = t4
User base				
Min		45	443	1187
Max		4012	12,055	47,766
Mean		219.31	1742.12	5823.99
Std Dev.		309.80	1892.12	6812.09
Developer base				
Min	1	1	1	1
Max	6	9	15	26
Mean	1.30	2.36	4.45	5.76
Std Dev.	0.88	1.60	3.82	5.09
Complexity				
Min		0.0019	0.0015	0.0020
Max		201.83	317.57	324.48
Mean		9.90	13.38	14.75
Std Dev.		26.94	37.144	40.49
Modularity				
Min		8	9	26
Max		496	612	617
Mean		85.46	133.04	154.70
Std Dev.		99.45	117.47	125.45
License type				
Min	1	1	1	1
Max	2	2	2	2
Mean	1.58	1.58	1.58	1.58
Std Dev.	0.49	0.49	0.49	0.49
Language translations				
Min		1	1	1
Max		5	19	35
Mean		1.23	2.15	4.75
Std Dev.		0.73	1.62	3.91
Responsibility assignment				
Min		0	0	0
Max		0.83	0.91	0.93
Mean		0.25	0.28	0.22
Std Dev.		0.18	0.22	0.2

5. Results

Table 1 shows summary statistics for the collected data for all variables. Table 2 shows sample correlation matrix for T = t2. The correlations were high enough to raise concerns of multicollinearity, which if uncorrected for may lead to inflated standard errors and, in worst case, inconsistent or unstable estimates (Greene, 2007). To ensure that multicollinearity was not an issue with the data, we conducted a formal multicollinearity test. We calculated Variance Inflation Factors (VIF) for each variable; the maximum value of VIF is 1.40, way below 10, which is usually considered the threshold above which multicollinearity may affect results (Neter et al., 1990).

Structural Equation Modeling (SEM) was used to analyze the data. The research models were tested using LISREL 8.80.[8] The syntax was written in SIMPLIS command language. Standardized path coefficients were calculated using LISREL path analysis.

Initial investigations indicated that the dependent variable and some of the independent variables were not normally distributed. In such case, linear regression analysis might yield biased and non interpretable parameter estimates (Gelman and Hill, 2007). Therefore, as suggested by Gelman and Hill (2007), we performed

[8] Several authors have suggested that PLS, as compared to LISREL, is computationally more efficient, makes minimal demands on sample's size and the distribution of the data. For this study, the research model was also tested using the SmartPLS software. As the results were similar to LISREL, which also provides additional statistics such as GFI, AGFI, CFI, and RMSEA, only LISREL results are reported.

Table 2
Sample correlation matrix (T = t2).

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
User base (1)	1.00									
Developer base (2)	−0.12	1.00								
Modularity (3)	−0.07	0.39	1.00							
Complexity (4)	0.07	−0.24	−0.22	1.00						
Language translation (5)	0.06	−0.08	0.01	−0.09	1.00					
Responsibility assignment (6)	−0.06	0.49	0.59	−0.33	0.17	1.00				
Market success (7)	−0.05	0.39	0.63	−0.38	−0.08	0.57	1.00			
Technical success (8)	0.00	0.09	0.32	−0.46	0.15	0.41	0.49	1.00		
License type (9)	0.03	−0.17	0.08	−0.17	0.25	0.14	0.07	0.16	1.00	
Release (10)	−0.02	0.02	0.08	0.03	0.05	0.01	0.00	−0.06	0.09	1.00

logarithmic transformations on non-normally distributed dependent and independent variables. Normality of the transformed variables was then confirmed using the Kolmogorov–Smirnov statistic (Mertler and Vannatta, 2005).

5.1. Structural model

For the first version, data for only the license type and the number of developers that contributed to a project can be collected and is available. Once the first version is released to the OSS community, potential users download it. Consequently, data for number of previous downloads could not exist. Furthermore, once the source code is available, we could obtain the values for other variables such as complexity and modularity of the source code and test their effect on project activity. But once the first version is released, these independent variables will impact the next version. Similarly, responsibility assignment and variables such as number of language translations, which was one for all the first version releases, could only be studied for next version onwards. Therefore, the model at the first version was run and tested only for popularity based on the two independent variables. From the results, it can be seen that both predictors: number of developers and license type play a significant role in predicting a project's popularity.

For other models, all independent variables were measured and tested. Model for T = t2 explained 86.1% of the variance in project popularity and 58.7% of the variance in developer activity. The model for T = t3 explained 86.4% of the variance in project popularity and 43.2% of the variance in developer activity. And, the model at T = t4 explained 55.5% of the variance in project popularity and 44.4% of the variance in developer activity. The values of CFI, GFI, and AGFI were over 0.90 for all the models, indicating a good fit. The results of the four different versions are summarized in Table 3.

6. Discussion

6.1. Popularity and developer activity

Hypothesis H1 proposed that OSS projects with higher technical success are more popular. The standardized parameters estimated for versions at T = t2/t3/t4 are 0.03/0.02/−0.08, respectively. Hence, the hypothesis that OSS projects with higher level of activity are more popular is not supported. However, contrary to our hypothesis, Stewart et al. (2005) tested the reverse relationship and suggested that project's popularity may impact the developer activity. It could be argued that a developer may perceive higher reputation within an OSS community if he contributes to a popular project.

6.2. License choice

Hypothesis H2a proposed that OSS projects that use a non-restrictive license are more popular than those that use a restrictive license. The values of the standardized parameters estimated for the four versions are 0.15/0.03/−0.02/−0.08. However, it is significant only at t1, indicating that the license restrictiveness plays significant role in the projects' popularity only for the first version of the OSS project. Hypothesis H2b hypothesized that OSS projects using a non-restrictive license have lower technical success than those using a restrictive license. The standardized parameter estimated for versions at T = t2/t3/t4 are 0.06/−0.18/−0.27. The estimated parameters are significant at t3/t4, and not at t2.

Project administrators must assess how the project's license will impact its likely popularity and developer activity. The preferences of individual developers for GPL, classified as a restrictive license, have been explained on a theoretical basis as a safeguard against private appropriation of collective efforts and a mechanism for lowering barriers to entry and maximizing the number of participants. Projects with non-restrictive licenses fail to combat developer's fear of losing control and they fear the possibility of commercial exploitation of their contributions by third parties (Stewart et al., 2005). This fear leads them to contribute lesser to such projects. Take the case of Netscape and its Mozilla project, as an example. In 1998, Netscape failed to get desirable response from developer community under the "Netscape Public License", which allowed Netscape to take pieces of the open source code and turn them back into a proprietary project again (Hammerly et al., 1999). However, changing the license to "Mozilla Public License", under which Netscape could not regain proprietary rights to modifications of the code, later, allowed Netscape to obtain higher response (Lee, 1999).

Our results indicate that restrictive licenses such as GPL, where the developers' perceive their contributions to be safe, are more appealing to developers. These findings are similar to the findings of Stewart et al. (2005). However, in our study, as we have differentiated among different stages of projects, the projects with restrictive licenses are more appealing in the later stages of development (t3/t4). The impact of license choice was insignificant for an earlier version (t2). This could be attributed to the number of developers associated with a project at different stages. This impact may be insignificant at initial stages because of the fact that the license is selected at the start of a project by the developers who started the project. However, this selection may conflict with the beliefs about protecting contributions of developers that join the project during later versions. And, if the license is not restrictive, the fear of the possible commercial exploitation by third parties (Stewart et al., 2005) may cause the developers not to contribute and even leave the project. Therefore, it is crucial for project administrators to select a license that encourages subsequent developers to contribute to OSS projects without having to worry about commercial exploitation.

Note that license choice was found to have a significant impact on project popularity for the first version only. This could be because when the first version of a project is made available to the public, only a limited amount of information is available about the project. For example, of the eight independent variables in this research, information about only the license choice and developer base is available. In the absence of other cues, potential users

Table 3
Path coefficients for all the models.

Structural path	Path coefficients				Conclusion
	T = t1	T = t2	T = t3	T = t4	
H1: Technical success → Market success		0.04	0.02	−0.04	Not supported
H2a: License type → Market success	0.15†	0.02	−0.06	−0.09	Supported at t1
H2b: License type → Technical success		0.09	−0.17*	−0.28*	Supported at t3, t4
H3a: User base → Market success		0.88*	0.88*	0.55*	Supported
H3b: Developer base → Market success	0.36*	−0.17*	−0.04	0.19*	Mixed results
H3c: Developer base → Technical success		0.47*	−0.07	−0.20†	Mixed results
H4: Language translations → Market success		0.13*	0.12*	0.23*	Supported
H5: Responsibility assignment → Technical success		0.19*	0.27*	0.27*	Supported
H6: Complexity → Technical success		−0.15*	−0.42*	−0.50*	Supported
H7: Modularity → Technical success		0.17†	0.28*	0.30*	Supported

† p-Value < 0.05.
* p-Value < 0.01.

employ only these cues to make a decision about the OSS project. The impact of license choice on project popularity was insignificant at all stages after t1. The insignificance may be because consumers do not wish to be bothered by the license choice when information on other extrinsic attributes is readily available. This, in a way, agrees with *vox populi* that most of the end users do not read the licensing agreements.

6.3. Size of user base and developer base

As hypothesized in H3a, OSS projects that have a larger previous user base are more popular. This was true at all stages. In H3b, it was hypothesized that OSS projects with larger developer base are more popular. The standardized parameter estimated for the four versions are 0.36/−0.17/−0.02/0.19. Except at t3, all parameters are significant. However, the direction of the relationship at t2/t3 is the opposite of what was expected. Lastly, in H3c, it was hypothesized that OSS projects with a larger developer base exhibit higher developer activity. The standardized parameter estimates are significant at t2/t4, although the direction is negative at t4. Thus, the number of developers has a positive impact on technical success only at t2.

The number of downloads for various projects exhibited a highly skewed distribution with a long tail to the right. Such skewed distributions are found in a wide range of phenomena such as incomes, and web site popularity. Such distributions are called Pareto power law distributions and are normalized using logarithm transformation. There are a number of explanations for such distributions, but the most relevant is likely to be the winner-takes-all nature of project popularity. As a project grows in popularity it becomes more attractive since there is greater likelihood of good documentation, knowledgeable support, further development work, and software tools that work with it. Similar results are indicated by the highly significant values of parameter estimates for the impact of size of user base on project popularity.

The pattern displayed by the parameter estimates resembles the sigmoidal curve, shown in Fig. 3, which has been used to explain diffusion of innovation among adopters. Rogers (1995) categorized adopters into five categories depending on the time of adoption. Those who adopt new innovations at early stages are called innovators, which in our context would refer to developers and administrators who start the software development project. These innovators influence others by word of mouth, discussion forums, or advertising. When projects are updated constantly, the projects become more active which leads to more popularity at early stages. This increased popularity also acts as an external influence, which attracts more users and developers. These influences lead to a period of rapid growth where more users and developers get involved with the project. Adopters getting involved at this stage are called early adopters and early majority. The high values of 0.88 of the parameter estimates at t2 and t3 reflect this stage. These points are reflected as points 'a' and 'b' in Fig. 3. When the latter two groups of adopters, late majority and laggards, start adopting the innovations, the slope of the curve falls down. This is reflected in the drop in the parameter estimate at t4 and the fall in the slope.

The important thing to understand is the 'critical mass', which is defined as the minimal number of adopters for the rate of adoption to be self sustaining (Schoder, 2000). Before the point of critical mass is reached, the number of adopters grows very slow and product success is not necessarily ensured. But once the critical mass is reached, many customers have already adopted the product and it is time for other customers to follow suit. Therefore, it is extremely important for project administrators to ensure that critical mass is achieved at the earliest possible to ensure success of the project. The earlier hypothesis revealed that keeping project activity high and selecting the appropriate license can help administrators to do the same.

Successful OSS projects require a much larger developer group than the core group to fix bugs (Mockus et al., 2002). In our data set, the average numbers of developers associated with the four versions were 1.6/2.5/4.6/5.5. The largest number of developers associated with a project was 26. This clearly indicates that while some projects that are very well publicized attract large number of developers, most OSS products are developed and maintained by a small number of developers. Further, our results found the impact of the size of developer base on popularity to be significant in the expected direction only in the first version. The relationship was also significant at t2 and t4, but in the reverse direction. No satisfactory explanation could be found for this behavior. However, Singh and Tan (2009) suggest that successful OSS projects need different types of developer participation. They suggested that successful OSS projects have to involve a handful of core developers to get the majority of coding done, while embracing all other developers and encouraging them to participate in creative thinking, discussions,

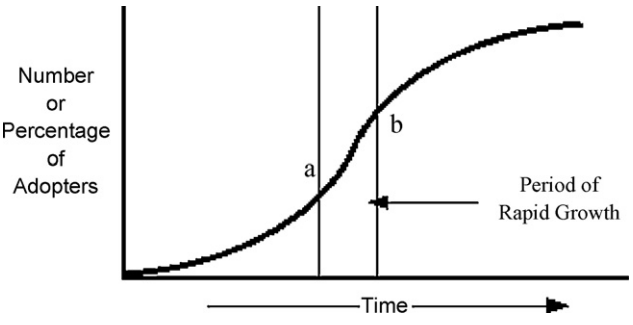

Fig. 3. Diffusion sigmoidal curve.

and providing feedbacks. Further work based on their suggestions can probably help find an explanation for this behavior.

The impact of the size of the developer base on technical success was insignificant at later versions. The hypothesis was that the size of the developer base will have a positive relationship with technical success at all stages. On the contrary, the parameter estimates at t3 and t4 displayed an inverse relationship indicating a decrease in technical success with increase in the number of developers. Previous research has indicated that most developers contributing at early stages are project initiators and do not represent more than 20% of the total number of developers (Mockus et al., 2002). When a project is started, the developers display a high level of enthusiasm as these are the developers who started the project to *scratch their itch* (Raymond, 2001). So, the higher number of active developers leads to higher activity. When the project grows old and new developers join the project, the level of enthusiasm falls down. Also as the team size increases, there is more effort and emphasis on managing and coordinating the team than enhancing the software, leading to a decrease in the level of activity. Another possibility is goal conflict between associated developers leading to *forking* of the project.

6.4. Language translations

H5 hypothesized that OSS projects with higher number of language translations are more popular. The standardized parameter estimates for versions at t2/t3/t4 are, 0.13/0.14/0.26, all significant, indicating that increasing language translations increases project popularity at each version.

Language is an important determinant of the effectiveness of the user interface. Studies have shown that even simple translation will increase the productivity of users of commercial systems (Bodley, 1993). A non-native language can pose obstacles for international business. Obviously a transaction cannot occur if the customer does not understand the language the web site is using. Take the case of Simputer (Kshetri, 2004), a handheld Internet appliance, as an example. Simputer is based on Linux, which has been adapted in various local languages. Simputer provides Internet and email access in local languages, micro-banking, speech recognition, and text-to-speech conversion and has been considered a huge success in India.

Our results suggest that project administrators can increase project popularity by increasing the available number of language translations as the project grows. However, adding language translations adds to the project's cost and man hours. Therefore, rather than arbitrarily adding languages, project administrators should make themselves aware of the target audience and then ensure that their language requirements are met. Furthermore, administrators may also have to abide by local rules and regulations. For example, in France and Spain, the availability of local versions is a requirement by law.

6.5. Responsibility assignment

H6 hypothesized that OSS projects that delegate more responsibility experience higher developer activity. The standardized parameters estimated for versions at t2/t3/t4 were 0.19/0.25/0.27, and were all significant. Clearly, assigning responsibility has a strong impact on technical success at each version.

Research on employee satisfaction has shown that assigning responsibility is the anchor for any satisfying job and retaining employees. On the other hand, it is believed that open source works because it gives developers freedom of choice and expression (Scacchi, 2004). Developers can choose the projects they work on and what to contribute, and therefore, the direction of their own technical growth. Our research supports the first premise that assigning responsibility leads to higher developer activity. We cite the case of Apache Group (AG), the informal organization of people responsible for guiding the development of the Apache HTTP Server Project. AG assigns the responsibility for bug fixes or feature additions to volunteers and has been able to get work done in an effective manner (Mockus et al., 2002). Project administrators, however, must be aware that they need to assign responsibility to someone who is capable and has the resources to do it. Assigning a job without this knowledge can lower technical success and contribute to developers' dissatisfaction.

6.6. Complexity and modularity

H7 hypothesized that OSS projects with lower complexity will experience higher developer activity. The negative sign of the estimated coefficients at all stages clearly indicate that as the complexity of a software project increases, technical success decreases. On the other hand, H8 hypothesized that OSS projects with higher modularity experience higher developer activity. This relationship was also significant at all stages. Our analysis seems to clearly highlight the need for keeping a project less complex and employing a modular approach to software production. Consistent with (Feller and Fitzgerald, 2002), the complexity of projects kept increasing as the projects progressed. This makes the project less attractive resulting in lower developer activity. On the other hand, increase in modularity led to an increase in technical success and helped attract contributions from developers.

Simple things are easy to understand and manage. For software to be simple, developers and project administrators need to monitor its complexity and modularity. As software projects progress, they become increasingly complex making it difficult to understand and manage them (Feller and Fitzgerald, 2002). Thus, if administrators want developers to contribute to their projects, they must make sure that, at least, the project's complexity is not the reason for developers to not contribute. At the same time, administrators need to make sure that the project is modular enough to harness the collective intellect of multiple developers working in a parallel in a distributed development environment.

6.7. Project release

To control for the impact of release type, we used the variable 'Release' in our analysis. The results show that release type has a significant impact on market success only for version t4 (standardized parameter value = 0.16). This suggests that OSS users prefer major releases over minor releases for older projects. This could be attributed to the perception that major releases of the projects are more stable and has lesser vulnerabilities. A new major release may still suffer from few more bugs, but it must still be better than a previous version so as to encourage migration to the newer version (Deprez et al., 2007). Its impact on market success was found to be insignificant for all other versions as well as on technical success for all versions.

7. Validity threats and implications

As in many empirical studies, there are certain aspects which can lead to threats to the validity of our study. Therefore, the findings should be interpreted in light of the limitations arising from those threats. One of the limitations is the reliance on SourceForge data only. While Sourceforge has data about a vast collection of OSS projects, it does not capture all OSS projects, which is the ultimate population of interest. While the sample size is quite large to ensure statistical validity, the choice of the sample frame may have some bearing on the outcomes of the study. SourceForge provides the largest publicly available database of open source projects, but

the utility of the metrics collected in its concurrent versioning system, as well as how consistently these metrics are computed across different open source projects maintained by different groups of core developers may be questionable. We attempted to alleviate this concern by relying solely on objective measures, such as size of user and developer base, complexity, and modularity measures.

Another limitation is that technical success has been measured through the number of commits. Literature shows that the number of commits for OSS projects is usually skewed, i.e., most commits take place in the early stages of projects (Hindle et al., 2008). However, as we have studied four different stages separately without combining them, it should not impact our results. Furthermore, Hindle et al. (2008) pointed that some projects accept a significant number of contributions from external sources and these are usually committed in one single large commit. In this study, we did not differentiate between types (importance or size) of commits, and we leave this level of analysis for future study. In addition, the notion of a developer, which was taken directly from the CVS logs, concerns construct validity. In some projects, people could be contributing code but not entering the modifications in the CVS logs, which sometimes are accessed by a limited number of developers or administrators. Therefore, the number of developers might actually be higher than the number reported. This fact is obviously problematic to check. Even though we have controlled for the impact of project release, a separate detailed study comparing all aspects for major and minor releases may provide additional useful results. Lastly, many of the projects have their own websites which may have their own followings, advertisements, mailing lists, governance mechanism, and download pages. This could also have an impact on the results of the four stage model.

As for future research, while the research model explained significant amount of variance in technical success and project popularity, more antecedents may improve its predictive power and provide deeper understanding. These antecedents may be based on finer granularity of variables, project age and characteristics, and types of audiences. For example, considering characteristics of licenses beyond only restrictiveness or considering more than two levels of restrictiveness may enhance our understanding of how licensing choice influences developers and users. It may also be necessary to distinguish between restrictiveness and familiarity of the license. As one of the oldest and most widely used open source license, GPL has high name recognition which may have influenced its appeal among users and developers.

Open Source advocates have always emphasized the benefits of free-will. Our results suggest that assigning responsibility to someone, the way it has been traditionally done, works better in terms of the level of activity. This seems contrary to popular belief and may have to be studied in detail. The study of the importance of the task assignment along with the skills and reputation of the developer could provide additional insights. Furthermore, it will be worthwhile to evaluate the moderating effects of other variables, such as the type of application, the operating environment, underlying programming language, and the types of audiences, on the research model.

Among the practical value of the results is the knowledge about how the independent variables influence users and developers. The independent variables are factors that can be controlled by the project administrators. Many of them can even be altered at future releases of the software. Project administrators need to recognize the need for these changes, provide facilitation in making the changes, and communicate to the user and developer communities so that they can reap the benefits. For example, one of the most interesting practical implications is that complexity, modularity, and responsibility assignment can be adjusted to enhance developer activity. More importantly, these can be adjusted in the current version to improve technical success and project popularity in future versions.

Our findings suggest that project administrators should select restrictive licenses, such as GPL, when starting a project to attract more contribution from developers. The impact may not be visible at early stages of the project, but it does impact both the number of developers contributing to the project and their level of activity at later stages. Note, however, that (Lerner and Tirole, 2002) suggested that this impact may not be as strong if the project administrators are considered well-trusted.

It is also important for project administrators to maintain high developer activity at early stages. The high level of activity acts as a marketing tool and attracts new users and developers. High activity can be maintained by constantly adding features and fixing bugs to the software. More importantly, project administrators must announce the updates and enhancements to the community in order to send signals of high activity. Additionally, in order to attract contributions from developers, administrators need to maintain low complexity and high modularity of the project. Note that in the extreme, high modularity can lead to greater complexity; therefore, administrators need to keep modularity within an acceptable range.

Contrary to developers' freedom of choice to work (Scacchi, 2004), assigning fixed responsibility developers leads to higher level of activity. While assignment is recommended, it should not be made arbitrarily; administrators must match the needs with developers' skills.

References

Bikhchandani, S., Hirshleifer, D., Welch, I., 1998. Learning from the behavior of others: conformity, fads, and informational cascasdes. Journal of Economic Perspectives 12 (3), 151–170.

Bodley, G., 1993. Design of Computer User Interfaces for Third World Users. Unpublished Master's Dissertation. University of Port Elizabeth.

Capiluppi, A., Lago, P., Morisio, M., 2003. Evidences in the evolution of OS projects through changelog analysis. In: Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on OSSE, vol. 1, pp. 9–22.

Chengalur-Smith, S., Sidorova, A., 2003. Success and survival of open-source projects: a population ecology perspective. In: Proceedings of the 24th ICIS.

Crowston, K., Howison, J., Annabi, H., 2006. Information systems success in free and open source software development: theory and measures. Software Process Improvement and Practice 11 (2), 123–148.

Darcy, D.P., Kemerer, C.F., Slaughter, S., Tomayko, J.E., 2005. The structural complexity of software an experimental test. IEEE TSE 31 (11).

Davis, F., Bagozzi, R., Warshaw, P., 1989. User acceptance of computer technology: a comparison of two theoretical models. Management Science 35, 982–1003.

Deprez, J.-C., Ruiz, J., Herraiz, I., Campos, C.G., 2007. Evaluation Report on Existing Tools and Existing F/OSS repositories Qualoss Sponsored Research Contract # 033547. Available from: www.qualoss.org/about/Progress/deliverables/WP1_Deliverable1.1.pdf (accessed 08.01.11).

English, R., Schweik, C., 2007. Identifying success and tragedy of FLOSS commons: a preliminary classification of Sourceforge. net projects. In: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development, IEEE Computer Society, p. 11.

Feller, J., Fitzgerald, B., 2002. Understanding Open Source Software Development. Addison-Wesley, London.

Gelman, A., Hill, J., 2007. Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press.

Gill, G., Kemerer, C., 1991. Cyclomatic complexity density and software maintenance productivity. IEEE TSE 17 (12), 1284–1288.

Gorla, N., Ramakrishnan, R., 1997. Effect of software structure attributes software development productivity. Journal of Systems and Software 36 (2), 191–199.

Greene, W.H., 2007. Econometric Analysis, 6th ed. Prentice Hall.

Grewal, R., Lilien, G.L., Mallapragada, G., 2006. Location, location, location: how network embeddedness affects project success in open source systems. Management Science 52 (7), 1043–1056.

Halstead, M.H., 1977. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., New York, NY.

Hammerly, J., Paquin, T., Walton, S., 1999. Freeing the Source: The Story of Mozilla in Open Sources: Voices from the Open Source Revolution. O'Reilly, Cambridge, MA, pp. 197–206.

Hindle, A., German, D., Holt, R., 2008. What do large commits tell us?: A taxonomical study of large commits. In: Proceedings of the 2008 International Workshop on Mining Software Repositories, ACM, New York, pp. 99–108.
Jones, T.C., 1986. Programming Productivity. McGraw-Hill, Inc., New York.
Kearney, J.K., Sedlmeyer, R.L., Thompson, W.B., Gray, M.A., Adler, M.A., 1986. Software complexity measurement. Communications of the ACM 29 (November (11)), 1044–1050.
Kemerer, C.F., 1995. Software complexity and software maintenance: a survey of empirical research. Annals of Software Engineering 1, 1–22.
Kemerer, C., Slaughter, S., 1997. Determinants of software maintenance profiles: an empirical investigation research and practice. Journal of Software Maintenance 9, 235–251.
Kshetri, N., 2004. Economics of linux adoption in developing countries. IEEE Software 1 (21), 74–81.
Lawton, G., 2002. The Great Giveaway. New Scientist., http://www.newscientist.com/article/mg 17323284.600-the-great-giveaway.html.
Lee, S., 1999. Open Source Software Licensing., http://cyber.law.harvard.edu/openlaw/gpl.pdf.
Lerner, J., Tirole, J., 2005. The scope of open source licensing. Journal of Law, Economics, and Organization 21 (April (1)), 20–56.
Lerner, J., Tirole, J., 2002. Some simple economics of open source. Journal of Industrial Economics 1 (2), 197–234.
Mansfield, E., Wagner, S., 1975. Organizational and strategic factors associated with probabilities of success in industrial R&D. Journal of Business 48 (2), 179–198.
Markus, M., Manville, B., Agres, C., 2000. What makes a virtual organization work? Sloan Management Review 42 (1), 13–26.
McConnell, J.D., 1968. The effect of pricing in an experimental setting. Journal of Applied Psychology 52, 331–334.
Mertler, C.A., Vannatta, R.A., 2005. Advanced and Multivariate Statistical Methods. Pyrczak Publishing, Los Angeles, CA.
Midha, V., Palvia, P., Singh, R., Kshetri, N., Spring 2010. Improving open source software maintenance. Journal of Computer Information Systems 2010, 81–90.
Mockus, A., Fielding, R., Herbsleb, J., 2002. Two case studies of open source software development: Apache and Mozilla. TOSEM 11 (3), 309–346.
Neter, J., Wasserman, W., Kutner, M.H., 1990. Applied Linear Statistical Models, 3rd ed. Irwin, Homewood, IL.
Olson, J.C., 1972. Cue Utilization in the Quality Perception Process: A Cognitive Model and an Empirical Test. Doctoral dissertation. Purdue University.
Parnas, D.L., 1972. On the criteria to be used in decomposing systems into modules. Communications of the ACM 15 (December (12)), 1053–1058.
Rai, A., Lang, S.S., Welker, R.B., 2002. Assessing the validity of IS success models: an empirical and theoretical analysis. Information Systems Research 13 (1), 50–69.
Ramanujan, S., Cooper, R., 1994. A human information processing approach to software maintenance. Omega 22 (2), 185–203.
Rana, Z., Khan, M., Shamail, S., 2006. A comparative study of spatial complexity metrics and their impact on maintenance effort. In: Emerging Technologies, 2006. ICET'06. International Conference on, pp. 714–718.
Raymond, E.S., 2001. The Cathedral & The Bazaar. O'Reilly.
Rilling, J., Klemola, T., 2003. Identifying comprehension bottlenecks using program slicing and cognitive complexity metrics. In: Proceedings of the 11th IWPC.
Rogers, E., 1995. Diffusion of Innovations, 4th ed. Free Press, NY.
Scacchi, W., 2004. Free and open source development practices in the game community. IEEE Software 21 (1), 59–66.
Schoder, D., 2000. Forecasting the success of telecommunication services in the presence of network effects. Information Economics and Policy 12, 181–200.
Schweik, C., Semenov, A., 2003. The institutional design of open source programming: implications for addressing complex public policy and management problems. First Monday 8 (1.).
Singh, P., Tan, Y., 2009. Stability and Efficiency of Communication Networks in Open Source Software Development. Working Paper.
Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G.L., 2002. Code quality analysis in open source software development. Information Systems Journal 12 (1), 43–60.
Stewart, K., Ammeter, A., Maruping, L., 2005. A preliminary analysis of the influences of licensing and organizational sponsorship on success in open source projects. In: Proceedings of the 38th HICSS.
Subramaniam, C., Sen, R., Nelson, M.L., 2009. Determinants of open source software project success: a longitudinal study. Decision Support Systems 2 (46), 576–585.
Torvalds, L., 1999. The Linux Edge. Open Sources: Voices from the Open Source Revolution. O'Reilly, Sebastopol, CA.
Vakratsas, D., Ambler, T., 1999. How advertising works: what do we really know? The Journal of Marketing 63 (January (1)), 26–43.
von Hippel, von Krogh, E.G., 2003. Open source software and the 'private-collective' innovation model: issues for organization science. Organization Science 14 (2), 209–225.
West, J., 2003. How open is open enough? Melding proprietary and open source platform strategies. Research Policy 32 (7), 1259–1285.
Wiggins, A., Crowston, K., 2010. Reclassifying success and tragedy in FLOSS projects. In: Proceedings of the OSS.
Wright, P.L., 1975. Consumer choice strategies: simplifying vs optimizing. Journal of Marketing Research 11, 60–67.
Wynn, D.E., 2003. Organizational structure of open source projects: a life cycle approach. In: 7th Annual Conference of the SAIS.
Zeithaml, V.A., 1988. Consumer perceptions of price quality, and value: a means-end model and synthesis of evidence. Journal of Marketing 52, 2–22.

Vishal Midha is an Assistant Professor of Information Systems in the Department of Computer Information Systems and Quantitative Methods at the University of Texas-Pan American. He has a PhD in information systems from the University of North Carolina at Greensboro. His current other research interests include open source software development, information privacy concerns, and internet frauds. He has published in Communications of AIS, International Journal of Electronic Commerce, Electronic Markets, Journal of CIS, and many national and international conferences including International Conference of Information Systems, Americas Conference on Information Systems, Decision Sciences Institute's Conference. Presently, he also serves as an Associate Editor in International Journal of Information Security and Privacy.

Prashant C. Palvia is the Joe Rosenthal Excellence Professor at the Bryan School of Business & Economics at the University of North Carolina at Greensboro. He received his PhD, MBA, and MS from the University of Minnesota and BS from the University of Delhi, India. He is the Editor in Chief of the Journal of Global Information Technology Management. His research interests include global information technology management, healthcare information technology, virtual teams, open source software, electronic commerce, media choice theory, and trust in exchange relationships. He has published 90 articles in journals such as MIS Quarterly, Communications of the ACM, Communications of the AIS, Information & Management, Decision Support Systems, and ACM Transactions on Database Systems, and over 165 conference articles. He has co-edited four books on global information technology management.