THE UNIVERSITY OF TEXAS AT SAN ANTONIO, COLLEGE OF BUSINESS

Working Paper SERIES

May 04, 2007

WP # 0013IS-296-2007

Virtual Organizational Learning in Open Source Software Development Projects

Yoris A. Au* ; Darrell Carpenter; Xiaogang Chen; Jan G. Clark

Department of Information Systems and Technology Management
College of Business, University of Texas at San Antonio
San Antonio, TX 78249, U.S.A.

UTSA COLLEGE OF BUSINESS KNOWLEDGE FOR A NEW WORLD

ONE UTSA CIRCLE
SAN ANTONIO, TEXAS 78249-0631
210 458-4317 | BUSINESS.UTSA.EDU

Virtual Organizational Learning in Open Source Software Development Projects

Yoris A. Au[*] ; Darrell Carpenter; Xiaogang Chen; Jan G. Clark

Department of Information Systems and Technology Management
College of Business, University of Texas at San Antonio
San Antonio, TX 78249, U.S.A.

Abstract

We studied the existence of virtual organizational learning in open source software (OSS) development projects. Specifically, our research focused on learning effects of OSS projects and factors that affect the learning process. The number and percentage of resolved bugs and bug resolution time of 118 SourceForge.net OSS projects were used to measure the learning effects. Projects were characterized by project type, number and experience of developers, number of bugs, and bug resolution time. Our results provide evidence of virtual organizational learning in OSS development projects.

Keywords: Virtual organizational learning; Organizational learning curve; Virtual organization; Open source software; Software development; Project performance

Acknowledgment:
The authors wish to thank the anonymous reviewers from the 18th Information Resources Management Association International Conference (IRMA2007) for useful suggestions on an early version of this paper, Greg Madey for providing access to SourceForge.net database, the participants in a seminar at the University of Texas at San Antonio in 2007 for their helpful comments, and the College of Business of the University of Texas at San Antonio for providing financial and technical support for this research.

[*] Corresponding author: Yoris A. Au, Information Systems and Technology Management, College of Business, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249; Phone: 1-210-458-6337; Fax: 1-210-458-6305; E-mail: yoris.au@utsa.edu.

Virtual Organizational Learning in Open Source Software Development Projects

1. Introduction

Open source software (OSS) development projects exhibit many of the characteristics that make virtual organizations successful, including self-governance, a powerful set of mutually reinforcing motivations, effective work structures and processes, and technology for communication and coordination [30]. Examples of thriving OSS projects include Linux (often considered a long-term challenge to Microsoft's operating system), Apache (which has more than 60 percent of the Web server market share), and Mozilla (whose Firefox Web browser is considered by many users to be superior to Microsoft's Internet Explorer).

Raymond [36] described the open source method of development as "a great babbling bazaar of differing agendas and approaches… out of which a stable and coherent system could seemingly emerge only by a succession of miracles." Although seemingly disorganized, and lacking monetary incentives, the bazaar development approach is characterized by design simplicity, team work, a visible product, and communication [42].

Researchers have studied different aspects of this particular form of virtual organization to identify the specific factors that make OSS development projects work. For example, Mockus et al. [31] conducted a case study on the Apache Web server and Mozilla Web browser projects to learn their development process characteristics. They found that projects based on a relatively small core of developers (10 to 15 people) could be geographically dispersed, yet communicate and function without conflict via a set of

implicit coordination mechanisms (i.e. informal email exchange). However, when the number of core developers exceeded 10-15 people, other explicit coordination mechanisms (i.e. code ownership policy) had to be adopted to maintain communication and reduce conflict.

In a related study, Huntley [21] attempted to explain the success of OSS projects using organizational learning effects. He maintained that the learning effects could be manifested in the decreased time required for fixing bugs. There were significant debugging differences in Apache versus Mozilla, with the attributing factor being project maturity, as opposed to other measurable factors such as project size or number of programmers. Debugging data from Apache and Mozilla were modeled according to the learning curve. As noted, the debugging process for Mozilla, an emerging project, was characterized as steady with predictable improvements. The results illustrate that the learning effects are indeed present in the Mozilla team.

Both Mockus et al. [31] and Huntley [21] focused their research on two OSS projects—Apache and Mozilla. While they pointed out significant differences between the two projects, there was no indication of which project was most characteristic of other OSS projects Our research seeks to extend and refine their works by including a much larger number of OSS development projects of varying size (in terms of the number of developers involved) and type (from simple file management software to complex enterprise software suite). Specifically, we included 118 OSS projects in our final sample set. By focusing on multiple projects of varying size and type, we were better able to characterize OSS projects overall. Our study was initiated to answer the following main research questions:

(1) Are learning effects universally present in OSS projects?

(2) What are the factors that affect the learning process?

Similar to Huntley [21], we use the number and percentage of resolved bugs and bug resolution time to measure the learning effects. However, we also look at how different project types, number of developers (project team size) and their experience, and the intensity of assigned bugs affect the learning rates. Data for this study were obtained from the SourceForge.net[1] database. Our study contributes to the information systems literature by providing empirical evidence of virtual organizational learning and of the factors that affect it.

2. Theoretical framework and hypotheses

We developed several hypotheses based on theories that relate to virtual organizational learning. Our first hypothesis seeks to show that organizational learning exists in OSS development projects. The subsequent hypotheses seek to explain the variation of learning rates observed across projects.

2.1 Organizational learning curves

Group learning curves were first observed in the 1940's during construction of ships and aircraft [46]. It was noted that the time required to build a ship or airplane typically decreased at a diminishing rate as more products were produced. The organizational learning curve is based on a combination of effort and learning. Moderating factors may include skill level, prior experience, motivation, and work complexity. An early

[1] Details on SourceForge.net's database are avalaible at http://zerlot.cse.nd.edu/mywiki/ ("SourceForge Research Data Archive: A Repository of FLOSS Research Data"). Christley and Madey [7] provide further descriptions of the SourceForge.net data set and discuss various data mining techniques that can be applied to the data.

discussion on organizational learning curves can be found in Wright [44], and reviews in Yelle [46] and Dutton and Thomas [11].

Although there is little empirical evidence about what contributes to the organizational learning curve phenomenon [11,12,28,46], researchers have suggested several factors: increased proficiency of individuals, greater standardization of procedures, improvements in scheduling, improvements in product design, better coordination, and division of labor and specialization [19,20,22,44]. Fiol and Lyles [15] postulate there are two levels of organizational learning: higher-level and lower-level. Higher-level learning focuses on re-defining the overall organizational strategy under ill-defined context. Examples of such learning include developing a new organizational culture and re-establishing organizational priorities [2]. Conversely, lower-level learning is focused on specific organizational behaviors and constraints within existing organizational rules. Minor managerial adjustments, improved problem-solving skills, and the development of formal rules are examples of lower-level learning [15]. This type of learning is primarily a process of repetition [8].

In OSS development projects, debugging can be considered as a means in which organizational experience is accumulated, thus helping to establish the software development learning curve [21]. As such, learning during the debugging process can be classified as lower-level learning since it often improves problem-solving skills by repeatedly requiring the developers to scan, review, and/or modify program code. It is expected that, as a development team gains experience with problem domains, techniques, technologies, and tools, it will exhibit the learning curve effect by decreasing, over time, the average time needed to resolve bugs. It is also expected that the team will

become more familiar with the debugging process, leading to more efficient debugging. Therefore, we hypothesize that:

H1: As the number of bugs resolved to date increases, the average bug resolution time decreases.

2.2 Cognitive capital and developer experience

Cognitive capital refers those resources that provide shared representations, interpretations and systems of meaning among parties [32]. It is comprised of both expertise and the knowledge about how to apply the expertise in solving a problem. Over time, people develop cognitive capital as they learn the skills, knowledge, specialized dialogue, and norms of the practice and interact with others who share the same practice [43]. Consequently, it is expected that individuals with longer tenure in a shared practice have greater cognitive capital and are thus better able to share their knowledge with others.

OSS developers can be concurrently involved in more than one project, allowing them a greater opportunity to work with other developers, learn about the norms of OSS development practice, and accumulate longer tenure and more experience. Overall, the greater the number of projects in which a developer is involved, the greater the developer expertise. This translates into larger cognitive capital, which the developer may share with other team members to help improve project team performance [23]. Under some circumstances, individuals may be unwilling to share their cognitive capital because they may fear the loss of power associated with unique knowledge [9,34]. However, this is not expected within the OSS community. Instead, the OSS community exhibits factors such as altruism, pro-sharing norms, and reciprocity [23] that are positively related to

knowledge transfer behavior. These same factors have also been identified as the major reasons why OSS developers voluntarily join the OSS community [37,39]. This leads us to the following hypothesis:

H2: Teams with more experienced developers resolve bugs faster.

2.3 Task ownership

Task ownership is a psychological state in which a task performer takes personal interest and responsibility for the task. The degree of task ownership can impact how the task is accomplished [10]. Researchers generally agree that task ownership improves team effectiveness [6] and facilitates individual learning in both traditional and computer-support collaborative learning environments [5,25]. For example, students exhibit a sense of individual accountability when their grade is based on individual efforts in a group project [25]. This also helps to eliminate free-riders and hitch-hikers [40].

The relationship between task ownership and individual/team performance can be explained by Goal-Setting Theory from organizational psychology. The theory maintains task ownership helps task performers clarify their task goals [29]. In turn, the clarified goal enables the performers to focus attention on goal-related activities, thus improving performance. However, goal clarity cannot be achieved if the task performer deems the task too difficult. Rasch and Tosi [35] validated Goal-Setting Theory in software development teams.

On SourceForge.net, when a bug is submitted into the tracking system, the administrator assigns the bug to a developer who may or may not accept the assignment (based on interest, time available, and/or ability). At the time of assignment, the administrator will also add comments about the expected bug turn-around-time. If the

developer rejects the assignment, the administrator must either find another developer willing to accept the bug assignment or it remains unassigned. Essentially, the bug assigning process is an ownership determination process. Once determined, the owner has full responsibility for the bug. We presume that as more bugs are assigned to specific developers, their average bug resolution time decreases. Therefore, we hypothesize that:

H3: There is an inverse relationship between increasing the percentage of bugs assigned to specific developers and average bug resolution time.

2.4 Project category

At the time this research was conducted, Sourceforge.net classified its projects into thirteen categories. These categories include Clustering, Database, Desktop, Development, Enterprise, Financial, Games, Hardware, Multimedia, Networking, Security, SysAdmin, and VoIP. We posit that these projects differ in other areas than just category. Project complexity and timeliness, among others, may also relate to significant differences among the project categories. Complexity can be defined in terms of the number of interconnected components and the degree of interdependency between these components [3]. It has been empirically proven to lengthen project time [18,45]. Timeliness refers to the implicit time (i.e., "deadline) pressure a project experiences. It has been found to cause teams to enhance their efforts on a given task [15,16], and thus leads to improved performance. We therefore hypothesize that:

H4: Different project categories have different average bug resolution times.

2.5 Project team size

Research on teams under traditional co-located situations suggests that the appropriate size of a team depends on the nature of the task [24]. It is also commonly

agreed that the association between team performance and team size has an inverted U-shape [33,41]. If a team is too small, it does not have sufficient human resources to share the workload. On the other hand, if a team is too large, coordination overhead becomes extremely high and social loafing becomes a real concern [27].

Similar findings have been reported in the software development domain. Brooks' Law indicates that the capability of a programming team does not necessarily improve as more developers join in the team. Instead, "adding manpower to a late software project makes it later" [4, p. 25]. When the number of developers in a team increases linearly, the potential communication paths increase exponentially. For instance, if N developers work on a project, the possible communication paths are $(N^2 - N)/2$ [4], increasing the chance of communication breakdown occurring within the project team.

The relationship between team size and team performance in OSS communities might exhibit a different pattern than that in the traditional software development team. This is because the communication structure of the OSS project team is different. An OSS project team is typically composed of two sub-groups of developers: core developers and code contributors [31,36]. Core developers are largely responsible for making critical decisions regarding project development (i.e., when to release the next version and whether or not to implement a new feature). In order to reach consensus on such decisions, intensive communication among the core developers is critical. Raymond [36] recommends that there be no more than three core developers per project.

Code contributors are the labor force for coding. They receive well-defined subtasks (i.e. bugs) from core developers, work on the subtasks independently, and then report finished tasks back to core developers. As a result, the communication structure within

an OSS project follows a star topology. The core developers are the central "hub," and all the contributors connect to and through the "hub". Raymond [36] recommends as many code contributors as possible. This is echoed by Huntley [21], who maintains that a well-controlled debugging process can be distributed among a large number of programmers without significantly increasing communication costs because most bugs are somewhat limited in scope and involve a small fraction of the code. We argue that different team sizes have different challenges. A large team may have coordination problems, whereas a small team may encounter resource problems, affecting the overall team performance. This leads us to the following hypothesis:

H5: Average bug resolution time varies among project team size.

3. Research Method

To study the constructs of interest, it was necessary to collect data on a wide variety of project-related measures. These measures included development status, rank, bugs reported, patches, feature requests, support requests, developer registrations and other data associated with an open source software project. We chose SourceForge.net as our data source. SourceForge.net hosts over 100,000 projects for Open Source Code. It provides a multitude of tools to support collaborative development and allows developers to register their projects at no charge. These properties make it equally attractive to both large and small development efforts. Furthermore, the number of projects, the wide variety in terms of size and expertise, and the availability of event data make it an ideal data source for this research.

3.1 Empirical model

Based on the production function and motivated by Argote et al. [1] and Huntley [21] we developed a log-log regression model with both qualitative and quantitative variables:

$$\ln MeanResTime_{it} = \alpha_0 + \alpha_1 \ln CumResBugs_{it} + \alpha_2 \ln AvgDevExp_i + \alpha_3 \ln PctAssignedBugs_{it} + \sum_{i=1}^{12} \beta_i \, ProjCat_i + + \sum_{j=1}^{3} \gamma_j \, ProjSize_{ij} + \varepsilon_{it}$$

……...…..… (1)

Where:

$MeanResTime_{it}$	=	Mean time to resolve the bugs of Project *i* reported in Week *t*
$CumResBugs_{it}$	=	Cumulative resolved bugs of Project *i*, including Week *t*
$AvgDevExp_i$	=	Average number of other projects each developer in Project *i* has worked on
$PctAssignedBugs_{it}$	=	Percentage of assigned bugs in Week *t* of Project *i*
$ProjCat_i$	=	Category of Project *i*
$ProjSize_i$	=	Size of Project *i*, measured in terms of the number of developers in the project (1-2 developers; 3-7 developers; 8-15 developers; >15 developers)

The descriptive statistics for the model variables are provided in Table 1.

INSERT TABLE 1 ABOUT HERE
---

4. Data collection

Data collection began by selecting representative projects for analysis. We identified the top 50 projects in each of SourceForge.net's 13 primary software categories. These

categories include Clustering, Database, Desktop, Development, Enterprise, Financial, Games, Hardware, Multimedia, Networking, Security, SysAdmin, and VoIP. Developers use these categories to describe their projects and help potential users and contributors find relevant projects. Because these categories were widely used for classification, they were determined to be the most appropriate method of ensuring a good cross-section of all open source projects that were selected for our sample.

The top 50 projects were identified based on two factors, development status and site rank. The first factor limited projects to those that had produced a production/stable version of the application. This was determined using SourceForge.net's development status field which lists project status as 1) planning stage, 2) pre-alpha, 3) alpha version, 4) beta version, 5) production/stable version, or 6) mature. This factor was necessary to ensure that "conceptual" projects with no event reports would not reduce the set of usable responses. Additionally, those projects that reached alpha or beta versions but had not yet produced production/stable versions needed to be excluded. In some cases, the developers were unresponsive to externally reported events (such as bug reports) and thus no inferences could be made from those projects.

The second factor for selection was SourceForge.net's internal ranking system. The ranking system used three sub-factors 1) traffic, 2) communication, and 3) development to determine an overall rank of projects. The traffic sub-factor included web traffic to the project's main SourceForge.net page, web traffic to all project sub-pages, and project-related file downloads. The communication sub-factor consisted of the number of opened or closed events, number of posts in the project's mailing lists, and number of posts in the project's discussion forums. The development sub-factor encompassed the number of

project commits made to SourceForge.net's proprietary versioning system, the age of the last file release, and how recently the project's administrators had logged on to the SourceForge.net site. This multi-factor ranking system has several desirable qualities that enhance the sample validity. One of the primary benefits of using the ranking system as a selection criterion is that older projects tend to drop in activity along several dimensions and thus ultimately drop in ranking. Using the rank criterion ensured that the projects selected reflect the current state of Open Source development. Based on the factors of development status and site rank, a "snapshot" of the top 50 projects in each category was collected on March 9, 2006. One category, VoIP, had only 47 projects that had developed a production/stable version of their product. This resulted in an initial identification of 647 projects for the potential sample pool.

The final data set was determined by applying three additional criteria necessary to produce a sample appropriate for testing the research hypotheses. These criteria include: 1) assigning projects registered in more than one of SourceForge.net's 13 software categories to their most appropriate category, 2) limiting projects to those that were at least two years old, and 3) limiting projects to those with a minimum of 100 bugs reported.

The first criterion was used to resolve data duplication issues when a project was listed in the SourceForge.net database as belonging to two or more software categories. For this study, a project was only included in its highest ranked category. For example, if a project was listed as the 5^{th} ranked project in the networking category and the 12^{th} ranked project in the clustering category, it was only included in the networking category. In one instance, a project was ranked as the top project in two categories. When

independently examined by two researchers, each researcher selected the same category as being most appropriate for this project. The project was therefore included in the category selected by the researchers. Within the sample, there were 51 projects listed in two categories, and 5 projects listed in 3 categories. No projects were listed in more than 3 categories. Removing the duplicate data reduced the potential sample size by 61 projects to 586 projects.

The minimum project duration of two years was established to select projects appropriate for comparison with Huntley [21]. As expected, many of the top ranked projects did not meet this longevity requirement. This is not surprising given that projects often experience a flurry of activity and a resulting rise in rank early in the project lifecycle. Many of these projects will suffer a severe reduction in activity and ranking once the initial project setup activity has subsided. The large size of the potential sample pool was intended to allow for a substantial loss of projects when this setup activity was culled from the data set. Applying the two year criterion reduced the sample size from 586 projects to 140 projects. It is important to note that although we set our minimum threshold for inclusion at 2 years or 104 weeks, all selected projects had at least 112 weeks of data. This enabled us to use the same 108 week horizontal-axis scale as Huntley [21] did in graphs for comparison.

Some projects had no usable event reports available despite their high ranking. We set a requirement of at least 100 reported bugs per project to address this issue and ensure that each project had sufficient bug data for analysis. While it might seem unusual for projects with no usable event reports to be ranked highly, there is a specific feature of SourceForge.net that makes this possible. SourceForge.net allows the developer to turn

off the external event reporting features or redirect users to an alternative event reporting system such as Bugzilla. However, despite the ability to turn off event reporting or redirect users to other reporting systems, the majority of projects chose to use the SourceForge.net site for these purposes. This is partially influenced by SourceForge.net's ranking system and its use of an activity statistic. Those projects using other sites for event reporting will generally have lower levels of activity. This is an intended consequence and SourceForge.net purposefully implements its ranking system to encourage developers to keep all project resources in one location. Applying the requirement for a minimum of 100 reported bugs reduced the sample by 22 projects to produce the final sample size of 118 projects.

SourceForge.net provides developers with tools for tracking four primary types of events: 1) bugs, 2) support requests, 3) patches, and 4) feature requests. We included data on all of these event types in our data set. These events are often referred to collectively as bug reports [21]. For continuity's sake, we will refer to all of these events collectively as bugs throughout the paper. Each bug can also be given a status such as open, closed, deleted, or pending. An important measure of organizational learning is a comparison of the ratio between reported bugs and closed bugs. After applying all project selection criteria our final pool of bugs across the 118 projects in the sample consisted of 91,745 reported bugs and 73,253 closed or resolved bugs. The data was then aggregated to produce weekly averages for each project. The result was a data set capturing 16,175 project-weeks of information across the 118 projects.

We also collected information about the developers associated with a given project. For each project we counted the number of registered developers. We also looked at each

individual developer to ascertain whether they were registered developers of other projects. This data was used to test our hypotheses that the ratio of developers to bugs is negatively related to bug resolution time and that the average number of projects that a developer participates in is negatively related to bug resolution time.

5. Data analysis and discussions

The distribution of projects and project-weeks across project categories are shown in Table 2. We ran an ordinary least squares regression based on the model in Equation (1) using SPSS 15.0. The Durbin-Watson statistic obtained from this regression was 1.528, which indicated there was a serial correlation problem with our data. This makes sense because our data set consists of repeated measurements of projects over time. Based on our sample size of 16,175 and model with 18 regressors, the critical values of the upper and lower bounds are $d_u = 1.967$ and $d_l = 1.576$ respectively [38]. To correct for this problem, we performed the Cochran-Orcutt procedure available in SPSS 15.0.

INSERT TABLE 2 ABOUT HERE

The model summary results and coefficient estimates, shown in Table 3, indicate that the model has a modest ability to predict average bug resolution times. This is not unexpected given the wide variability among open source development projects. However, the analysis is well suited for the intended purpose and provides valuable insight into the effects of the various hypothesized predictor variables as described below.

INSERT TABLE 3 ABOUT HERE

5.1 Learning curve effect

The literature suggests that the cumulative number of bugs resolved to date will have an effect on the current average bug resolution time [21]. The negative coefficient for *CumBugsResolved* (Table 3) indicates that average bug resolution time decreases as the cumulative number of bugs resolved increases, providing support for our hypothesis H1: As the number of bugs resolved to date increases, the average bug resolution time decreases. This indicates the presence of a learning curve effect.

In addition to learning curve effect, we also investigated the presence of adaptive learning in the projects in our sample by examining the ratio of cumulative resolved bugs to cumulative reported bugs. We plotted a graph of project effectiveness to show the effect of adaptive learning. The graph (Figure 1) indicates that there was an adaptive learning process, but the process varied based on the number of developers. In particular, projects with 1-2 developers learned faster but became less effective over time. Projects with >15 developers demonstrated the best efficiency over time, followed by projects with 3-7 developers. It is interesting to note that the variability of efficiency decreased substantially as the number of developers increased.

INSERT FIGURE 1 ABOUT HERE

5.2 Effect of developer experience

H2 states that teams with more experienced developers resolve bugs faster. The negative coefficient for developer experience supports this hypothesis. Average bug resolution time did decrease as developer experience increased. For this study, we measured experience as the number of projects in which a developer was registered. Based on this metric, the finding is consistent with previous research that suggests that teams comprised of more experienced developers were more effective [14]. We speculate that developers who worked on multiple projects learned new coding techniques and other "best practices" from those projects and conveyed that knowledge to other teams to which they belonged [23]. As a result, each team benefits from the developer's involvement with other teams.

5.3 Effect of bug assignment (task ownership)

H3 states there is an inverse relationship between increasing the percentage of bugs assigned to specific developers and average bug resolution time. The results support this hypothesis with a negative coefficient for *CumPctBugsAssigned* (Table 3*)*. This finding is consistent with previous research that indicated task assignment or ownership is a mitigating factor in reducing project risk [13]. The specific project risk in this case is that without task ownership project developers might simply avoid resolving bugs that had not been assigned to them.

5.4 Effect of project category

It was hypothesized that different project categories have different bug resolution times (H4). To study this effect, we divided the projects into the 13 primary categories used to classify projects on SourceForge.net. We coded the project categories using

binary dummy variables [26]. The analysis compared projects in the Clustering category (which is the reference category) to the other 12 categories. The positive coefficients for project category (Table 3) show that most project types had significantly higher bug resolution times than the reference category. Only SysAdmin projects had a moderately lower average bug resolution time than Clustering projects. The average bug resolution time for Hardware projects was not significantly different from that of Clustering projects. The other 10 project categories have coefficients that are different in value and statistically significant. This indicates that each of these project categories had a different average bug resolution times, supporting our hypothesis H4.

5.5 Effect of number of developers or team size

We also hypothesized that bug resolution time varies among project team size (H5). To test the impact of project size (number of developers), we divided the projects into four categories consisting of 1-2, 3-7, 8-15, and >15 developers. We used dummy variables to represent these developer categories in the model [26]. The results indicate that all developer categories had lower resolution times than the reference category of 1-2 developers. The coefficients indicate a curvilinear pattern with projects utilizing 3-7 developers showing improvement over those with 1-2 developers and projects with 8-15 developers having the lowest average bug resolution time. The average resolution time then increased for projects with greater than 15 developers. The likely causes for this curvilinear pattern in the average bug resolution time include communication complexity, organizational complexity, management effectiveness and project complexity as the size of the project changes.

6. Conclusions

In a knowledge-based economy, organizational learning is one of the most critical aspects that an organization needs to acquire and develop to achieve outstanding and sustained performance. As the trend towards virtual organizations continues to increase with advances in the Internet and telecommunication technologies, virtual organizational learning has become more and more important as well. However, although much has been published regarding virtual organizations, there is a lack of empirical research that shed light on the existence of learning that this type of organization can experience. Our study contributes to the body of literature by providing empirical evidence of virtual organizational learning in a large number of open source software (OSS) development projects.

Our results suggest that both adaptive learning and organizational learning curves were observed in the OSS projects. We also found that OSS development project performance was influenced by the number of developers on the project team, the amount of experience that those developers possessed, project category, and the percentage of bugs assigned to a specific person. In addition, the results indicate that although smaller teams learned faster, they suffered from greater variability in efficiency. Projects with a large numbers of developers (>15) demonstrated the best efficiency over time.

Our research has several limitations. The sampling technique used for this effort may not be ideal for a more in-depth analysis of the factors identified as affecting OSS team performance. Researchers should explore other selection criteria such as matched samples that may be more suited for studying specific factors. For example, a researcher interested in studying team size may benefit by selecting sample data that is matched in

terms of project category characteristics such as complexity and time pressure. The sample can also be controlled for developer experience and percentage of bugs assigned. Additionally, the experience measure used in this study is a static value based on team membership at the time of data collection. A more detailed analysis that tracks team membership by week may provide greater insight into the effect of developer experience. Another approach that may prove useful when measuring developer experience is to identify core developers and those who are merely code contributors. This technique may allow researchers to better understand optimal core team size and its effect on performance. Finally, developer experience measures may benefit from the inclusion of items such as formal education or years of programming experience.

While our research suggests that project performance does vary based on project type, our research design does not allow us to explicitly test the impact of complexity, timeliness concerns or other variables that the literature suggests as underlying causes of performance differences. One approach to overcoming this limitation may be to survey project team members to determine the relative complexity and time pressures associated with specific projects. Another suggested tactic is to triangulate the results of multiple methods or metrics that may be associated with systematic differences in performance among project types. Examples of suitable methods for studying this phenomenon include secondary data analysis, survey research, and observation. Metrics that may hold insight into performance include bug resolution time, lines of code, and number of code modules.

The results of this study suggest a number of avenues for additional inquiry. First, additional research is needed to further determine optimal team size for both the core

team and ancillary code developers. This research might also explore the tendency of projects to subdivide into functional units as they grow beyond a certain point. If support for a subdivision phenomenon is found, researchers should attempt to identify the underlying causes of such subdivision. While the literature strongly suggests communication complexity as the primary determinant of optimal team size, there may be other factors that have yet to be identified. Researchers should also seek to identify the range of team sizes where subdivision normally occurs.

Future research may also contribute to the body of knowledge on OSS performance by identifying other factors that affect bug resolution time. This is particularly important in light of the modest prediction ability of the current model. Although there is a great deal of variance expected among individual projects, other systematic factors may exist. Along with identifying other factors that affect bug resolution time, researchers should also seek to identify other salient measures of OSS project effectiveness. Exploration of efficiency and effectiveness measures should include both qualitative and quantitative items of importance to each project stakeholder.

References

[1] L. Argote, S.L. Beckman, D. Epple, The persistence and transfer of learning in industrial settings, Management Science 36 (2), 1990, pp. 140-154.

[2] C. Argyris, D. A. Schŏn, Organizational Learning, Addison-Wesley, Readings, MA, 1978.

[3] D. Baccarini, The concept of project complexity – a review, International Journal of Project Management 12 (4), 1996, pp. 201-204.

[4] F. Brooks, The Mythical Man-Month, Addison-Wesley, 1975.

[5] J.S. Brown, A. Collins, P. Duguid, Situated cognition and the culture of learning, Educational Researcher 18, 1989, pp. 32-42

[6] M.A. Campion, G. J. Medsker, A. C. Higgs, Relations between work group characteristics and effectiveness: implications for designing effective work groups, Personnel Psychology 46, 1993, pp. 823-847.

[7] S. Christley, G. Madey, Analysis of activity in the open source software development communicty, in: R.H. Sprague, Jr., Proceedings of the 40th Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos, CA, 2007.

[8] R.M. Cyert, J. G. March, A Behavioral Theory of the Firm, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[9] T.H. Davenport, L. Prusak, L. Working Knowledge: How Organizations Manage What They Know, Harvard Business School Press, Boston, 1998.

[10] V. Druskat, A. Pescosolido, The content of effective teamwork mental models in self-managing teams: ownership, learning and heedful interrelating, Human Relations 55, 2002, pp. 283-314.

[11] J.M. Dutton, A. Thomas, Treating progress functions as a managerial opportunity, Academy of Management Review 9, 1984, pp. 235-247.

[12] D. Epple, L. Argote, R. Devadas, Organizational learning curves: a method for investigating intra-plant transfer of knowledge acquired through learning by doing, Organization Science 2 (1), 1991, pp. 58-70.

[13] J.M. Erikson, R. Evaristo, Risk factors in distributed projects, in: R.H. Sprague, Jr., Proceedings of the 39th Hawaii International Conference on System Sciences, IEEE, Los Alamitos, CA, 2006.

[14] S. Faraj, L. Sproull, Coordinating Expertise in Software Development Teams, Management Science 46 (12), 2000, pp. 1554-1568.

[15] C.M. Fiol, M.A. Lyles, Organizational learning, Academy of Management Review 10 (4), 1985, pp. 803-813.

[16] C. Gersick, Time and transition in work teams: toward a new model of group development, Academy of Management Journal 31, 1988, pp. 9–41.

[17] D. Gladstein, Groups in context: a model of task group effectiveness, Administrative Science Quarterly 29, 1984, pp. 499–518.

[18] A. Griffin, The effect of project and process characteristics on product development cycle, Journal of Marketing Research 34 (1), 1997, pp. 24-35.

[19] R.H. Hayes, S.C. Wheelwright, Restoring Our Competitive Edge: Competing through Manufacturing, Wiley, New York, NY 1984.

[20] W.Z. Hirsch, Manufacturing progress functions, Review of Economics and Statistics 34, 1952, pp. 143-155.

[21] C.L. Huntley, Organizational learning in open-source software projects: an analysis of debugging data. IEEE Transactions on Engineering Management 50 (4), 2003, pp. 485-493.

[22] P.L. Joskow, N.L. Rose, The effects of technological change, experience and environmental regulation on the construction cost of coal-burning generating units, Rand Journal of Economics 16, 1985, pp. 1-27.

[23] A. Kankanhalli, B.C.Y. Tan, K. Wei, Contributing knowledge to electronic knowledge repositories: an empirical investigation, MIS Quarterly 29 (1), 2005, pp. 113-143.

[24] R. Klimoski, R. Jones, Team Effectiveness and Decision Making in Organizations, Pfeiffer & Co, San Francisco, CA, 1995.

[25] P. Kirschner, J-W. Strijbos, K. Kreijins, P. J. Beers, Designing electronic collaborative learning environments, Educational Technology Research & Development 52, 2004, pp. 47–66.

[26] M.H. Kutner, C.J. Nachtscheim, J. Neter, W. Li, Applied Linear Statistical Models, 5th ed., McGraw-Hill Irwin, Boston, MA, 2005.

[27] B. Latane, K. Williams, S. Harkins, Many hands make light the work: the causes and consequences of social loafing, Journal of Personality and Social Psychology 37 (6), 1979, pp. 822-832.

[28] M.B. Lieberman, The learning curve and pricing in the chemical processing industries, Rand Journal of Economics 15, 1984, pp. 213-228.

[29] E.A. Locke, G.P. Latham, Building a practically useful theory of goal setting and task motivation, American Psychologist 57 (2), 2002, pp. 705-717.

[30] M.L. Markus, B. Manville B, C.E. Agres, What makes a virtual organization work? Sloan Management Review 42 (1), 2000, pp. 13-26.

[31] A. Mockus, R.T. Fielding, J.D. Herbsleb, Two case studies of open source software development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology 11 (3), 2002, pp. 309-346.

[32] J. Nahapiet, S. Ghoshal, Social capital, intellectual capital, and the organizational advantage, Academy of Management Review 23 (2), 1998, pp. 242-266.

[33] V.F. Nieva, E.A. Fleishman, A. Reick, Team dimensions: their identity, their measurement, and their relationships, U.S. Army Research Institute for the Behavioral and Social Science, Washington, D.C., Research Note 85 (12), 1985.

[34] W.J. Orlikowski, Learning from notes: organizational issues in groupware implementation, Information Society 9 (3), 1993, pp. 237-251.

[35] R.H. Rasch, H.L. Tosi, Factors affecting software developers' performance: an integrated approach, MIS Quarterly (Sept.1992), pp. 395-413.

[36] E.S. Raymond, The cathedral and the bazaar, First Monday 3(3) (1998). <www.firstmonday.org/issues/issue3_3/raymond/index.html> (accessed 04.27.07).

[37] J.A. Roberts, I-H Hann, S.A. Slaughter, Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the Apache projects, Management Science 52 (7), 2006, pp. 984-999.

[38] N.E. Savin, K.J. White, The Durbin-Watson test for serial correlation with extreme sample sizes or many regressors, *Econometrica* 45 (8), 1977, pp. 1989-1996.

[39] K.S. Shah, Motivation, governance, and the viability of hybrid forms in open source software development, Management Science 52 (7), 2006, pp. 1000-1014.

[40] R.E. Slavin, Cooperative learning in teams: state of the art, Educational Psychologist 15, 1980, pp. 93-111.

[41] I.D. Steiner, Group Process and Productivity, Academic Press, New York, NY, 1972.

[42] C. Wagner, Breaking the knowledge acquisition bottleneck through conversational knowledge management, Information Resources Management Journal 19 (1), 2006, pp. 70-83.

[43] M.M. Wasko, S. Faraj, Why should I share? Examining social capital and knowledge contribution in electronic networks of practice, MIS Quarterly 29 (1), 2005, pp. 35-57.

[44] T.P. Wright, Factors affecting the cost of airplanes, Journal of the Aeronautical Sciences 3, 1936, pp. 122-128.

[45] W. Xia, G. Lee, Grasping the complexity of IS development projects, Communications of the ACM 47 (5), 2004, pp. 68-74.

[46] L.E. Yelle, The learning curve: historical review and comprehensive survey, Decision Sciences, 10, 1979, pp. 302-328.

Table 1. Descriptive Statistics

Variable	N	Range	Min	Max	Mean	Std. Dev.
MeanResolutionTimeWeek *t* (DV)	16175	1546.06 1	0.00	1546.061	29.882	92.399
CumBugsResolved	16175	6148	0.00	6148	474.67	878.772
DeveloperExperience	16175	6.67	1.00	7.67	2.458	1.280
CumPctBugsAssigned	16175	1.00	0.00	1.00	.558	.291
ProjectCategory						
Enterprise	16175	1	0	1	.06	.240
Desktop	16175	1	0	1	.07	.250
SysAdmin	16175	1	0	1	.11	.311
Financial	16175	1	0	1	.01	.111
Development	16175	1	0	1	.31	.461
Games	16175	1	0	1	.08	.266
Security	16175	1	0	1	.08	.275
Multimedia	16175	1	0	1	.02	.122
Database	16175	1	0	1	.07	.259
Hardware	16175	1	0	1	.05	.217
Networking	16175	1	0	1	.09	.288
VoIP	16175	1	0	1	.01	.109
DeveloperCategory						
3-7 Developers	16175	1	0	1	.26	.441
8-15 Developers	16175	1	0	1	.31	.464
> 15 Developers	16175	1	0	1	.33	.470

Table 2. Distributions of Projects and Project-Weeks

Project Category	Number of Projects	Project-Weeks
Clustering	6	730
Networking	8	1481
Multimedia	4	244
Hardware	7	800
VoIP	3	194
SysAdmin	12	1751
Games	9	1244
Security	9	1333
Development	32	4943
Database	9	1171
Enterprise	8	994
Desktop	9	1088
Financial	3	202

Project Size	Projects in Category
1-2 Developers	13
3-7 Developers	36
8-15 Developers	36
> 15 Developers	33

Table 3. Model Summary Results and Coefficient Estimates

	Unstandardized Coefficients		
	β	Std. Error	Significance
(Constant)	.307	.184	0.095*
CumBugsResolved	-0.080	0.008	0.000***
DeveloperExperience	-0.659	0.075	0.000***
CumPctBugsAssigned	-0.063	0.006	0.000***
P (Rho)	0.249	0.008	0.000***
ProjectCategory			
SysAdmin	-0.280	0.166	0.092*
Hardware	-0.088	0.192	0.646
Security	.592	0.173	0.001***
Enterprise	0.665	0.180	0.000***
Desktop	1.061	0.179	0.000***
Financial	1.109	0.292	0.000***
Development	1.233	0.145	0.000***
Database	1.869	0.174	0.000***
Networking	2.121	0.167	0.000***
Games	2.203	0.172	0.000***
Multimedia	2.405	0.282	0.000***
VoIP	5.774	0.320	0.000***
DeveloperCategory			
3-7 Developers	-1.410	0.120	0.000***
8-15 Developers	-1.709	0.115	0.000***
> 15 Developers	-1.484	0.119	0.000***
Model Summary			
R^2	0.101		
Adjusted R^2	0.100		
Model Std. Error	3.428		
df Regression	18		
df Residual	16154		
df Total	16174		
Significance	0.000***		

Note: Significance level = *p < 0.10; ** p < 0.05; *** p < 0.01.

Figure 1. Debugging Effectiveness of OSS Projects