Diversity in Software Development Routines are Attractive: A Preliminary Analysis of GitHub Repositories

Full Paper

William N. Robinson
Georgia State University
wrobinson@gsu.edu

Tianjie Deng
University of Denver
tdeng1@student.gsu.edu

Abstract

Free, libre, open-source software projects (FLOSS) are known for their chaotic development style and unique collaboration model. How does such chaotic development produce high quality software and attract users and developers? To provide insight into this conundrum, this study explores the roles of diversity and change in design routines. It investigates the relationship between routine diversity and change on project attraction to users and developers. Various sequence-mining techniques such as motif analysis and hidden Markov models (HMM) are applied to examine design routines of 88 FLOSS projects on GitHub.com. Regression analysis reveals that development processes with high routine-diversity and relatively low change-magnitude attract more users and developers.

Keywords

open-source software, design routines, project attraction.

Introduction

This study looks at design routines in the context of open source software development, showing a relationship between design routine diversity and changes with project attraction.

Open Source Software Development

Open source software projects (FLOSS) are known for their chaotic development style (Mockus et al. 2002). Several significant characteristics of FLOSS development are the following:

- Work is self-assigned: contributors choose what they want to undertake (Crowston et al. 2007; Crowston et al. 2008).
- There is a lack of coordination mechanisms, which are observed in traditional development settings—there are few formal "plans, system-level design, schedules, and defined processes" (Crowston et al. 2007; Herbsleb et al. 1999; Mockus et al. 2002, p.310).
- Multiple different processes are performed by contributors simultaneously (Christley et al. 2007).

Those chaotic processes produce high quality software. A Six Sigma or CMM perspective would not agree that chaos produces good quality, consistently. This raises the question: how does a seemingly chaotic system produce good software? As it turns out, FLOSS does have technical and managerial mechanisms for coordination(Mockus et al. 2002). These mechanisms affect the way developers produce software—the design routines they use, when they change their routines, and how dramatic those changes are. Design routines affect software qualities, including how valued the software is. In this study, we found that some FLOSS projects moderately vary their design processes over time, which may help them to maintain the interest of their developers and users in the face of a fast-changing environment.

Design Routines

According to Gaskin *et. al.*, "a *design routine* as a sequence of (design) tasks, which transform some representational inputs into a set of material and representational outputs, leading ultimately to a generation of a design artifact that offers a set of functions for a community of users. Unlike other routines, such as payroll, that perform highly standardized tasks with clearly defined inputs, outputs and transformation rules, design routines are fluid. They often deal with unknown inputs and outputs due to changing requirements that result from learning and environmental volatility" (Gaskin et al. 2011).

In the context of open source software development, design routines are the activities that take place in a programmer's editor (e.g., Eclipse), the repository (e.g., GitHub), and the associated project management tools (e.g., Atlassian). These different contexts represent different levels of design routines. Within an editor, programming patterns and classic Design Patterns are applied(Gamma et al. 1994). Within a repository, coordination and community actions take place, such as forking (i.e., copying) a project to begin a new project variant(Duc et al. 2014). Within project management tools, developers coordinate and plan long-term actions and conduct multi-project management. The study described herein is concerned with the middle level of abstraction—the repository activities.

This study

Design routine research can be difficult because practices often differ from stated actions(Feldman 2000). Consequently, research studies typically analyze a single site, manually encoding mid-level design routines over a year of data, for example. Larger multi-site, multiyear studies review design routines at an abstract, strategic level. The study described herein is a multi-site, multiyear analysis of design routines within repository activities—the data are detailed events over 88 projects. We draw the conclusion that higher routine diversity implies attractiveness, while dramatic changes in routines may adversely affect its attractiveness. This result is consistent with previous research that attempts to reconcile the seemingly inconsistent conclusions of related research: (1) change is good, and (2) stability is good. The result drawn here is most consistent with Klarner *et al.*—regular, moderate change in design routines is good (Klarner et al. 2012).

In support of our research on design routines, we apply a process theory methodology (Van de Ven 2007). Data mining, especially sequence mining, provides the means to acquire and analyze the thousands of design routine sequences from hundreds of projects (Abbott 1990).

Next, we present related research to describe the concepts of our variance model, which is summarized in the following section. Subsequently, we describe our multi-site, multiyear study. The final section presents a brief discussion.

Theoretic Background

Definitions

The literature on routines uses some key terms, mostly associated with variations, in a variety of ways. Therefore, we first define some common terms, which we rely on in our analysis. First, we consider variation within a type, of types, and configurations of types. Page used the term diversity to define these three concepts (Page 2010):

- Diversity within a type, or **variation**. This refers to differences in the amount of some attribute or characteristic, such as the height of giraffes.
- **Diversity** of types and kinds, or species in biological systems. This refers to differences in kind, such as the different types of foods kept in a refrigerator.
- Diversity of **composition**. This refers to differences in how the types are arranged. Examples include recipes and molecules.

We will use Page's three kinds of differences: (1) variation within a type, (2) diversity of types, and (3) compositions. These definitions can be applied to the context of routines:

1. *Routine variation* occurs when the arguments (values) differ between routines of the same type.

2. *Routine diversity* occurs when different types of routines observed.
3. *Routine composition diversity* occurs when different configurations of routines types are observed.

These definitions build on each other. For example, in analyzing a design history, one might observe routine composition diversity (different configurations) with great routine diversity (many different types) with great routine variation (where each routine type was instantiated with unique arguments). These differences can be considered from a combinatorics perspective, where total design space is the cross product of all routine arguments, routine types, and their configurations. We will use the generic term, r*outine diversity*, to mean all three kinds of diversity, except when the context requires us to be more precise.

Entropy is another term that often arises in the analysis of design routines. Shannon's information entropy definition is most widely applied(Shannon 2001); it is calculated as $H = - \Sigma P(x_i) \ln P(x_i)$ where $P(x_i)$ is the probability of event x_i. We rely on Shannon's classic entropy definition, which is the amount of uncertainty in a variable. We will refer to these basic definitions, as we review related theory on routines.

Requisite Variety Theory

Organizational routines have been considered from a theoretical perspective. Their variety derives from what we term the *routine composition diversity*—that is, the differences of configurations, types, routine arguments as the human-computer system executes.

From a theoretical perspective, Ross Ashby formulated the law of Requisite Variety, which states "when the variety or complexity of the environment exceeds the capacity of a system (like an organization) to create the corresponding variety of answers, the environment will dominate and ultimately destroy that system" (Ashby 1956). This formulation compares an organization's variety of responses to it environment. To retain control, the organization must have sufficient variety to respond to the environment conditions. In our terms, if the environment has a great diversity of conditions and the organization has a lower diversity of responses, then the organizational actions will be relatively simplistic and ineffective. Therefore, effective organizations must have high routine diversity. In practice, there is little concern that designers have inadequate routines to control systems. However, this theoretical view shows the importance of routine diversity. We anticipate that FLOSS projects with higher routine diversity and some change routines will better be able to address the needs imposed by their dynamic environments.

Routines

Organizational routines have been empirically studied. Pentland *et. al.*, for example, studied changing routines. They found that change depended "… on the experience level of the humans. In particular, automation can increase the variety of approval routines when it occurs in conjunction with less experienced users…. In other situations, where the boundaries between steps are less rigidly structured and enforced, it seems plausible that variation, selection, and retention of microlevel patterns could initiate changes in the higher-level patterns. " (Pentland et al. 2011). They went on to speculate that, "If the mechanism of change is variation and selective retention, as hypothesized by Feldman and Pentland (2003), then greater variety is a prerequisite for change. Variation has long been recognized as a foundation for learning in general (Campbell 1965, Weick 1979) and for learning in routines in particular (Levitt and March 1988)." In summary, from their analysis of invoice processing in four organizations, Pentland *et. al.* note that routines change over time, in part because of less experience users, but also because of the allowed variability by the system and potential for exploration by users. This environment is like FLOSS because of the collaborative workflow environment leads to changing routines. However, FLOSS is more like their speculative, less rigidly structured environment, in which lower-level patterns (e.g., programming patterns) lead to changes in higher-level patterns (e.g., repository patterns). Pentland *et. al.*, provides a variety of reasons to anticipate design routine changes in FLOSS development.

Given the occurrence of changing routines, Salvato finds that ordinary activities are more likely to cause routine changes, than management intervention. An analysis of the design process for a home-furnishing firm revealed five kinds of changes to routines. Salvato's data mostly "portrays an organization whose core routines and capabilities develop more as a result of everyday, mundane activities than of managerial cognition" (Salvato 2009). The everyday techniques discovered by Salvator are consistent purposeful activities recommended for innovation (Jantsch 1967). They are consistent with FLOSS in that distributed, loosely coordinated members perform, and *change*, different routines simultaneously (Christley et al. 2007).

As a final illustrative empirical study, consider Klarner and Raisch's analysis of rhythms of strategic change in insurance companies as identified from annual reports (Klarner et al. 2012). In particular, c*hange frequency* refers to strategic changes. This is a high-level, abstract view of firms over nine years. They discovered four distinct rhythms of change. They found that "Companies that change regularly outperform those that change irregularly" (Klarner et al. 2012). Moreover, a "stability periods' initially positive effect on subsequent change gradually turns negative the longer the stability period lasts" (Klarner et al. 2012). It's worth noting that Klarner and Raisch's found that where there is high environmental dynamism, the temporarily switching strategy also had a positive effect on performance. FLOSS too has sought regularized change, now commonly associated with scrum, but also older methods like Microsoft's sync-and-stabilize (Ambler et al. 2012). Consequently, we anticipate routine diversity and changing routines in FLOSS.

The stability-change paradox in organizations has raised long-lasting debates (Feldman et al. 2003; Leana et al. 2000; March 1991). One group of scholars has advocated the benefits of fast-paced change in organizations (Adler et al. 1999; Brown et al. 1997; Burgelman et al. 2007; D'Aveni et al. 2007; Hannan et al. 1977; Hannan et al. 1984). There are two key supporting arguments: (1) change can prevent organizations from being too inflexibility; and (2) fast-paced change can establish routines for change (Adler et al. 1999; Nelson et al. 2009). Conversely, other scholars argued for the benefits of stability (Dierickx et al. 1989; Nelson et al. 2009). Some of them point out that periods of stability are needed to establish organizational routines, which require time to develop(March 1981). Another reason is that fast-paced change can result in information overload, which may have adverse effect on firm performance (Dierickx et al. 1989; Hambrick et al. 2005). By considering periods of change, Klarner and Raisch's research addresses some of the inconsistency between these different positions. In summary, regular change in routines improves a firm's performance. In dynamic environments, bouts of additional of change may be helpful. In this context, much FLOSS development is considered a dynamic environment. Thus, we anticipate that regular, and occasionally irregular, change in routines improves FLOSS development on some performance measures.

Open Source Routines

Through observations, interviews, and analysis of repository histories an interesting view of open source development emerges. First, it is worth noting that open source development does have managers, who specify policies of software development(Crowston et al. 2005; Mockus et al. 2002; Scacchi 2004; Scacchi et al. 2006). Such policies can be strictly enforced by imposing a workflow over the development tools—for example, as provided in IBM TeamConcert. Nevertheless, developers do vary from the specified policies—changing routines is common practice(Feldman 2000). For example, code forking is discouraged because it creates a variety of problems (e.g., divergent and redundant code)[1]. Yet, forking is a common practice, which solves problems such as schedule constraints, technical variations, and organizational differences(Duc et al. 2014). More generally, dependencies among development artifacts require coordination. This includes the coordination of module development, because of interface dependencies. It also include ancillary artifact coordination, such as test cases, design documents, user manual, FAQs, *etc.*, all of which are consistent with the code base, ideally. While these dependencies can be mitigated through a good technical architecture, often the developers must address problems through adapting their design routines(Mockus et al. 2002). This experimental adaptation and learning of routines is consistent with organizational research on routines (Levitt et al. 1988; Pentland et al. 2011; Salvato 2009).

[1] A code fork is a copy (a clone) of a code repository that has subsequent development independent of the original project (Robles et al. 2012).

Direct analysis of design routine activities in open source is limited. Lindberg conducted a case study on the Rubinius open source project to investigate the co-evolution relationship between coding practice and communities(Lindberg 2013). A review of 18 months of repository event data revealed a positive relationship between activity entropy and new forks. The study used entropy to measure the variety of coding practice structures (i.e., design routines), and used the number of new forks as proxy for inflow of new developers to the community. "To describe the evolution of coding practices, we measure the distribution of event types (i.e., ratio of push vs. pull vs. issue events) for each month of the 18 month period. We also measure the entropy of activity distributions in each time period....Understanding entropy is important, because it is an indication of the variety of practice structures in a given time period. A period dominated by a single activity (e.g. PushEvents) will have low entropy, whereas a period characterized by diverse activities will have high entropy." Open source development appears to be similar to other design processes, where design routines are diverse and changing. More study is needed; however, it appears that as the complexity of the task or team increases, the diversity of the design routines increase.

Moderating change

The prior discussion emphasizes changing routine diversity. However, not all kinds of change are good. There is some evidence that process consistency improves performance. This view is not necessarily inconsistent with the need for change.

Consider the philosophies of Six Sigma and the capability and maturity model (CMM) as applied to software development. As an organization improves on the CMM scale, there appears to be a "decrease in variability of schedule and cost performance"(Diaz et al. 1997), higher product quality, and increased development effort(Harter et al. 2000). The best CMM organizations seek to reduce process variation so as to provide statistical predictability, and thereby improve performance(Ahern et al. 2004). Reduced process variation does not mean limiting routine diversity. Each new development effort necessitates new *routine diversity,* differing from prior projects. Moreover, routine diversity can be regularized, as suggest by (Klarner et al. 2012). Thus, the micro-level patterns can change, even regularly, while the macro-level measures of quality, schedule, and effort remain consistent.

Six Sigma shares the CMM philosophy of reducing process variability(Murugappan et al. 2003; Schroeder et al. 2008). In Six Sigma, process variation is defined as deviation from process mean. Building on this concept, in their 1999 paper, Frei and his colleagues investigated the relationship between service process variation and firm performance in retail banking industry(Frei et al. 1999). They found that process variation was negatively related to firm performance. To reduce variability, they identify natural causes and special causes of variation and then attempt to mitigate the special causes—for example, fixing quality failures by lengthening schedules, which reduce rework and thereby improve quality(Murugappan et al. 2003). Reduced variability in project measures, such as quality, schedule, and effort, determine if the process has achieved its goals(Montgomery et al. 2008). Thus, *routine diversity* is consistent with philosophies of Six Sigma of CMM. The micro-level patterns of regular design-routine change, and in dynamic environments, occasional irregular change, can produce consistently the macro-level quality of good performance for a firm.

Project Attractiveness

FLOSS projects need to attract users and developers to keep a FLOSS project active and successful (Arakji et al. 2007; Koch 2004; Krishnamurthy 2002; Von Krogh et al. 2003). Important success factors include, developer motivation and interest to participant (Bonaccorsi et al. 2003; Crowston et al. 2002; Fang et al. 2009; Hertel et al. 2003; Krishnamurthy 2006; Roberts et al. 2006), and user interest(Subramaniam et al. 2009). Projects also have a self-reinforcing effect of attractiveness (Santos et al. 2013). Santos *et. al.* also confirmed the following factors that form users' and developers' perception on FLOSS projects' attractiveness: contextual factors of the project (e.g. license choice and application domain), visibility of the project, and the work activities performed towards software maintenance and improvement(Santos et al. 2013). The study herein uses fork rate and star rate to directly measure project attractiveness.

Model Development and Hypotheses

Having introduced related research conceptualizing and supporting routine diversity and change, we now present our model and hypotheses.

Relationship between Routine Diversity and Project Attraction

Routine diversity in FLOSS development attracts participation in these ways:

1. It accommodates change in complex systems, e.g., (Klarner et al. 2012; Page 2010).
2. It enhances innovation, e.g., (Jantsch 1967; Salvato 2009).
3. It enhances developer contributions, e.g., (Lindberg 2013).
4. It enhances learning, which is an important motivation of FLOSS participation (Bonaccorsi et al. 2003; Lerner et al. 2002).

Therefore, a FLOSS project with more routine diversity is likely to attract more users and developers. We use fork rate (daily numbers of new forks of a project) and star rate (daily number of new stars of a project), to measure project attraction to users and developers. A user or a developer can create a fork of a repository to experiment with changes to the repository, without affecting the original repository. A user or developer can also star a project to show his or her appreciation to the repository. Therefore, daily number of forks and daily number of stars show how much attraction a project obtained from users and developers. This leads to our first two hypotheses:

Hypothesis 1a: Deign routine diversity is positively associated with fork rate.

Hypothesis 1b: Design routine diversity is positively associated with star rate.

Relationship between Routine Change and Project Attraction

A regular rhythm of moderate change in routines is associated with good firm performance, whereas more random or dramatic change is not(Klarner et al. 2012). Dramatic changes in design or design routines increases design and development complexity, which in turn hinders participation(Cant et al. 1995; Robbins et al. 1996; Subramanyam et al. 2003). Therefore, a FLOSS project with more dramatic routine changes is likely to attract fewer users and developers. This leads to our second proposition.

Hypothesis 2a: Design routine change magnitude is negatively associated with fork rate.

Hypothesis 2b: Design routine change magnitude is negatively associated with star rate.

Figure 1 presents the propositions in the proposed research model.

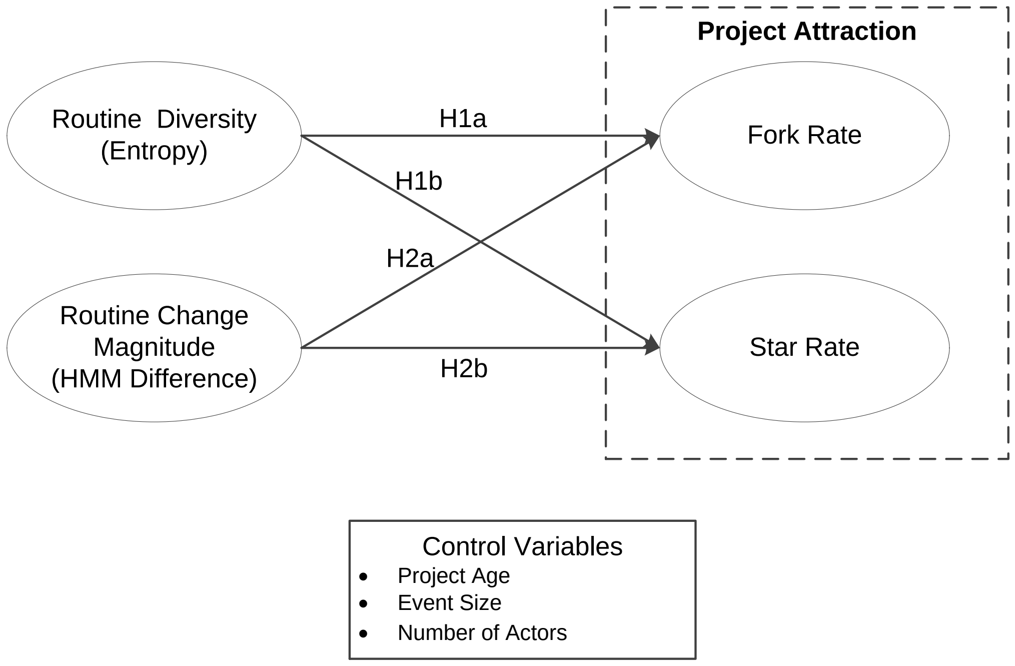

Figure 1 Research Model

Research Design

Data Collection

We collected data from 103 FLOSS projects from GitHub, the most popular open-source code repository site. Founded in 2008, GitHub had over 3 million users and over 5 million repositories as of January 2013. To obtain a sample with variation along the dependent variables, we used a stratified sampling strategy to sample projects with different level of popularity. The GitHub metrics, *number of stars* and *number of forks*, are proxies for the level of popularity to users and developers. We selected 30 projects from each of the following sets:

1. >= 10,000 stars and >= 1,000 forks
2. 5,000 >= stars < 10,000, and 750 >= forks < 1,000
3. 1,000 >= stars < 5,000 and 500 >= forks < 750
4. 1,000 > stars and 250 > forks and in Java

We distinguished Java (in set 4) to investigate if language plays a role in projects' development patterns. (Most GitHub projects are scripting languages, like JScript, rather than traditionally complied languages.) The initial 120 projects were reduced to the final 88 because some projects lacked sufficient data.

Data Preparation

Each project is represented as a sequence of development activities. Of the 18 Git repository activities, we focused on six, which are most closely associated with development teamwork, as shown in Table 5 of the Appendix.

The activity sequences were organized into *work motifs*, (aka design routines). The concept of motif has been used in different domains, such as biochemistry, to represent "a recurring theme or pattern" (Altarawy et al. 2009). Our work motif is a recurring sequential pattern of development activities.

The work motifs are operationalized in a rule-base system, but are summarized here as regular expressions:

1. (IssueEvent | PullRequestEvent) .*
2. (Reopen (of #1)) .*

As indicated above, a work motif begins with either an IssueEvent or PullRequestEvent, followed by events that reference the initiating event. This reflects the way that developers think about their work—either as a new issue (#1) or as the reopening of a prior issue (#2). When either an IssueEvent or PullRequestEvent is reopened, it is consider a new instance of the second motif pattern.

We obtained a total of 92,988 work motifs. (Summary statistics of these motifs is provided in Table 6 in the Appendix.) In our initial set of work motifs, different instantiations of the same work motif present different levels of collaboration. For example, in an IssueEvent->CommentEvent->PushEvent work motif, the level of collaboration would be different between an instantiation in which the same actor performed all the activities, and an instantiation in which three different actors performed the three activities. Distributed cognition (DCog) theory, which considers how team members collaborate through social distribution and structural distribution, provides a basis to interpret these data(Hutchins et al. 1996). We used this theory to differentiate instantiations of work motifs based on collaboration attributes. For each project, we applied *k*-means to classify work motifs into 50 clusters based on 8 DCog dimensions: type of open event, open actor and close actor (same or different), duration, final state (open or closed), result (merged or not), number of unique actors, number of comments, and total number of events in a work motif. Thus, each work motif has an associated DCog cluster, or type, ranging from 1 – 50.

Measurement

After the initial preparation, the 88 project datasets were processed to provide values for the variables presented next.

Dependent variables

Many different measurements of FLOSS success have been utilized. Crowston and his colleagues have conducted three studies to define OSS success and measures (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006). They categorized measurements of FLOSS performance and success into three types: measurements concerning the process, project output, and outcomes for project members. Among them, we are focusing on the measurements that are related to project attraction. For example, user and developer satisfaction(Crowston et al. 2006; Lee et al. 2009) has been used commonly. Another commonly used measurement is user interest, often operationalized as number of downloads(Crowston et al. 2002; Grewal et al. 2006; Méndez-Durón et al. 2009), number of page reviews(Crowston et al. 2002), and number of subscribers(Sen et al. 2012; Subramaniam et al. 2009). Because users' interest can be change over time, some studies have measured the change of number of subscribers over time(Stewart et al. 2006).

In this study, we use *fork rate (daily numbers of new forks)* and *star rate (daily number of new stars),* as dependent variable to measure project attraction to users and developers. Forks show the popularity of a project with users or developers. Creating a star for a project, allows the user to "create a bookmark for easier access (to the project) and show appreciation to the repository maintainer for their work". In either case, a star shows the user's interest or satisfaction. Many of GitHub's repository rankings depend on the number of stars a repository has. For example, repositories can be sorted and searched based on their star count.

Independent variables

Two variable constructs measure process diversity and process change:

1. Entropy, measures the uncertainty of different activity types; it indicates the diversity of the design routines.
2. ΔHMM, measures the magnitude of change in transition probabilities; it indicates the degree of design routine change.

Average entropy is an independent variable for each project. We used Shannon-Wiener index (Shannon 2001), a commonly used diversity index, to measure the average process diversity in a project. Shannon's index has been used to calculate entropy, which has been defined as a transversal distribution of activities(Gabadinho et al. 2011). In our context, entropy is the distribution of different work motifs. A data window dominated by a single type of work motif will have low entropy, whereas a window characterized by diverse work motifs will have high entropy. After some preliminary analysis, we choose four-weeks for our data window size—it contains sufficient data and represents a common unit of work for open source development methodologies. Entropy is calculated for each window to provide a sequence of project entropies over time—their average is used for the regression.

Average Hidden Markov Model (HMM) difference is an independent variable for each project. Given data containing sequences, a common task is to find transition probabilities. That is, given an observed event A, what is the probably that the next event observed with be B or C? A hidden Markov model (HMM) can solve this problem by building a probability model from observed event sequences(Rabiner 1989). For each data window, an HMM (λ_i) is created representing the work motif transition probabilities. Consider two HMMs in sequence, λ_1 and λ_2. Model differencing characterizes the change: $d\lambda/dt = (\lambda_2 - \lambda_1) / (t_2 - t_1)$. This gives the magnitude of change in the transition probabilities, for a sequence of data windows —their average is used for the regression. This is generally interpreted as behavioral change over time(Robinson et al. 2014); here, it indicates the magnitude of design routine change.

Control variable

A variety of characteristics can affect the popularity of a FLOSS project. For example, the longer a project has existed, the more likely that it will be widely known and consequently obtain forks and stars. Number of contributors may also increase forks and stars. Therefore, we control for project age, average number of actors per motif, and average number of events per motif. The control variable definitions are provided in Table 1 and the dataset descriptive statistics is provided in Table 2.

Table 1 Variables and Description

	Variable Name	**Definition**
Dependent Variables	*Fork Rate*	Number of forks per day
	Star Rate	Number of stars per day
Independent Variables	*Entropy*	Average uncertainty of event distribution in a given time period
	ΔHMM	Average HMM difference of the project
Control Variables	*Project Age*	Project age
	Event Size	Average number of events per motif of the project
	Number of Actors	Average number of actors per motif of the project

Table 2 Description Statistics

Variable	Mean	Std. Dev.	Min	Max
1. Star Rate	8.22	0.68	0.68	32.4
2. Fork Rate	1.54	0.22	0.22	5.51
3. Event Size	4.42	1.06	1.06	11.17
4. Number of Actors	2.22	1.03	1.03	3.53
5. Project Age (day)	1,058.2	210	210	2342
6. ΔHMM	1.61	0.13	0.13	3.1
7. Entropy	1.83	0.18	0.18	3.52

Data Analysis

We applied ordinary least-squares (OLS) regression to estimate the two dependent variables: *daily forks* and *daily stars*. Before completing our analysis, we checked assumptions of the multiple linear regression model for the ordinary least squares method (i.e., normality and multicollinearity among independent variables). Diagnostic checks on residuals were conducted to ensure the assumptions of normal distribution of residuals are not violated. We also examined the multicollinearity among explanatory variables.

Research Results

Table 3 presents the regression results. As can be observed, number of daily forks, is positively and significantly associated at the 0.001 significance level with entropy. Number of daily forks, is negatively and significantly associated at the 0.01 significance level with ΔHMM. A similar pattern arises from the other dependent variable, number of daily stars; it is positively and significantly associated at the 0.001 significance level with entropy. Number of daily stars, however, was not significantly associated with ΔHMM. Thus, there is strong statistical support for design routine diversity and change with outcomes of project attraction. Additionally, variables that significantly affect daily forks and daily stars include event size, number of actors, and project age. A summary of results obtained for the proposed hypotheses is presented in Table 4.

Table 3 Estimation Results

Variable	Fork Rate (adj R^2=0.53)	Stars Rate (adj R^2=0.57)
Constant	1.60***	6.89**
Event Size	-0.23*	0.05*
Number of Actors	0.87*	4.89**
Project Age	0.00***	-0.01***
Design Routine Diversity (Entropy)	0.50***	2.71***
Design Routine Change Magnitude (ΔHMM)	-0.38**	-0.84

Note: *: $P \le 0.05$. **: $P \le 0.01$. ***: $P \le 0.001$

Table 4 Results of Hypotheses

Hypothesis	Estimation results
H1a	Supported
H1b	Supported
H2a	Supported
H2b	Not supported

Discussion

In the context of FLOSS development routines, entropy measures routine diversity, while ΔHMM measures the magnitude of change. The results reveal that when a project has a diversified set of routines, it attracts more developers and users. On the other hand, when a project experiences dramatic routines changes, it becomes less appealing to developers. Furthermore, it would seem that active developers are more sensitive to both measures, while more passive people (e.g., users and occasional developers) are less sensitive. People that only occasionally interact with a project may not even notice the change. Taken to the extreme, an uncontrolled, chaotic project with wild swings in routine diversity may be recognized by active developers as a failure, which then causes a drop in daily forks; yet, those less aware or affected by the apparent pending doom do not significantly alter their daily stars. This line of reasoning rationalizes the findings of Table 4. These findings will contribute to both researchers and practitioners. With the insights provided by this study, FLOSS participants can better steer their projects to attract more developer participation. The study also contributes to literature both in FLOSS development and routines.

To extend the discussion, we illustrate a speculation about projects types according to the two dimensions of entropy and ΔHMM in Figure 2. Chaotic, floundering projects, commonly termed thrashing (or death march(Yourdon 2003)), are illustrated in the upper right of Figure 2. A maintenance project, with its simple, occasional updates, is the opposite. Regular moderate change indicates an innovative, successful project(Klarner et al. 2012). While frequent changes that have little lasting effect on routine diversity may indicate ineffective management interventions(Salvato 2009). Future research is necessary to understand how these dimensions affect project type.

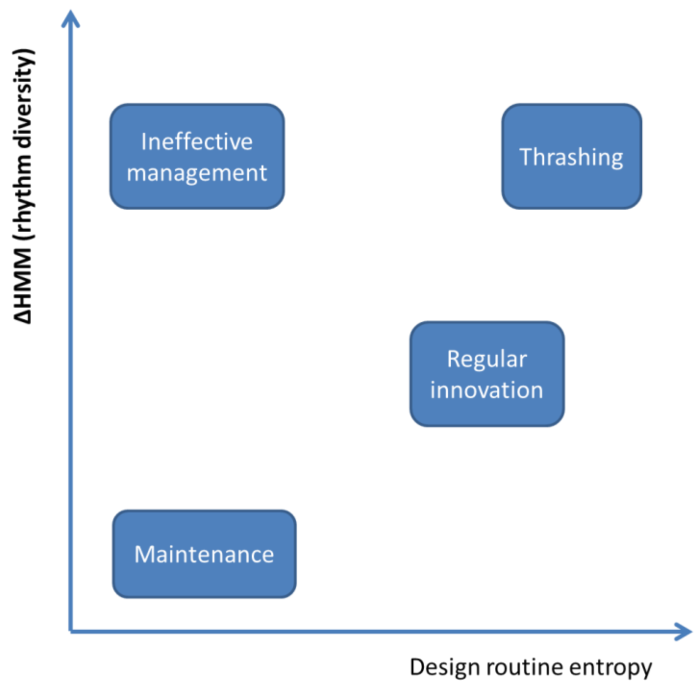

Figure 2 Hypothesized projects types.

REFERENCES

Abbott, A. "A primer on sequence methods," *Organization Science* (1:4) 1990, pp 375-392.

Adler, P.S., Goldoftas, B., and Levine, D.I. "Flexibility versus efficiency? A case study of model changeovers in the Toyota production system," *Organization science* (10:1) 1999, pp 43-68.

Ahern, D.M., Clouse, A., and Turner, R. *CMMI distilled: a practical introduction to integrated process improvement* Addison-Wesley Professional, 2004.

Altarawy, D., Ismail, M.A., and Ghanem, S.M. "MProfiler: A profile-based method for dna motif discovery," in: *Pattern Recognition in Bioinformatics*, Springer, 2009, pp. 13-23.

Ambler, S., and Lines, M. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise* IBM Press, 2012.

Arakji, R.Y., and Lang, K.R. "Digital consumer networks and producer-consumer collaboration: Innovation and product development in the digital entertainment industry," System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, IEEE, 2007, pp. 211c-211c.

Ashby, W.R. "An introduction to cybernetics," *An introduction to cybernetics.*) 1956.

Bonaccorsi, A., and Rossi, C. "Why open source software can succeed," *Research policy* (32:7) 2003, pp 1243-1258.

Brown, S.L., and Eisenhardt, K.M. "The art of continuous change: Linking complexity theory and time-paced evolution in relentlessly shifting organizations," *Administrative science quarterly*) 1997, pp 1-34.

Burgelman, R.A., and Grove, A.S. "Let chaos reign, then rein in chaos—repeatedly: Managing strategic dynamics for corporate longevity," *Strategic management journal* (28:10) 2007, pp 965-979.

Cant, S., Jeffery, D.R., and Henderson-Sellers, B. "A conceptual model of cognitive complexity of elements of the programming process," *Information and Software Technology* (37:7) 1995, pp 351-362.

Christley, S., and Madey, G. "Analysis of activity in the open source software development community," System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, IEEE, 2007, pp. 166b-166b.

Crowston, K., Annabi, H., and Howison, J. "Defining open source software project success," *ICIS 2003 Proceedings*) 2003, p 28.

Crowston, K., Annabi, H., Howison, J., and Masango, C. "Effective work practices for software engineering: free/libre open source software development," Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research, ACM, 2004, pp. 18-26.

Crowston, K., and Howison, J. "The social structure of free and open source software development," *First Monday* (10:2) 2005.

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

Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., and Howison, J. "Self-organization of teams for free/libre open source software development," *Information and Software Technology* (49:6), 6// 2007, pp 564-575.

Crowston, K., and Scozzi, B. "Open source software projects as virtual organisations: competency rallying for software development," IET, 2002, pp. 3-17.

Crowston, K., and Scozzi, B. "Bug fixing practices within free/libre open source software development teams,") 2008.

D'Aveni, R.A., and Gunther, R. "Hypercompetition. Managing the dynamics of strategic maneuvering," in: *Das Summa Summarum des Management*, Springer, 2007, pp. 83-93.

Diaz, M., and Sligo, J. "How software process improvement helped Motorola," *IEEE software* (14:5) 1997, pp 75-81.

Dierickx, I., and Cool, K. "Asset stock accumulation and sustainability of competitive advantage," *Management science* (35:12) 1989, pp 1504-1511.

Duc, A.N., Mockus, A., Hackbarth, R., and Palframan, J. "Forking and coordination in multi-platform development: a case study," Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2014, p. 59.

Fang, Y., and Neufeld, D. "Understanding sustained participation in open source software projects," *Journal of Management Information Systems* (25:4) 2009, pp 9-50.

Feldman, M.S. "Organizational routines as a source of continuous change," *Organization science* (11:6) 2000, pp 611-629.

Feldman, M.S., and Pentland, B.T. "Reconceptualizing organizational routines as a source of flexibility and change," *Administrative Science Quarterly* (48:1) 2003, pp 94-118.

Frei, F.X., Kalakota, R., Leone, A.J., and Marx, L.M. "Process variation as a determinant of bank performance: evidence from the retail banking study," *Management Science* (45:9) 1999, pp 1210-1220.

Gabadinho, A., Ritschard, G., Mueller, N.S., and Studer, M. "Analyzing and visualizing state sequences in R with TraMineR," *Journal of Statistical Software* (40:4) 2011, pp 1-37.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns Elements of Reusable Object-Oriented Software* Addison-Wesley, 1994.

Gaskin, J., Thummadi, V., Lyytinen, K., and Yoo, Y. "Digital Technology and the variation in design routines: a sequence analysis of four design processes,") 2011.

Grewal, R., Lilien, G.L., and Mallapragada, G. "Location, location, location: How network embeddedness affects project success in open source systems," *Management science* (52:7) 2006, p 1043.

Hambrick, D.C., Finkelstein, S., and Mooney, A.C. "Executive job demands: New insights for explaining strategic decisions and leader behaviors," *Academy of management review* (30:3) 2005, pp 472-491.

Hannan, M.T., and Freeman, J. "The population ecology of organizations," *American journal of sociology*) 1977, pp 929-964.

Hannan, M.T., and Freeman, J. "Structural inertia and organizational change," *American sociological review*) 1984, pp 149-164.

Harter, D.E., Krishnan, M.S., and Slaughter, S.A. "Effects of process maturity on quality, cycle time, and effort in software product development," *Management Science* (46:4) 2000, pp 451-466.

Herbsleb, J.D., and Grinter, R.E. "Splitting the organization and integrating the code: Conway's law revisited," Proceedings of the 21st International Conference on Software Engineering, ACM, 1999, pp. 85-95.

Hertel, G., Niedner, S., and Herrmann, S. "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel," *Research policy* (32:7) 2003, pp 1159-1177.

Hutchins, E., and Klausen, T. "Distributed cognition in an airline cockpit," *Cognition and communication at work*) 1996, pp 15-34.

Jantsch, E. *Technological forecasting in perspective: A framework for technological forecasting, its technique and organisation; a description of activities and an annotated bibliography* Organisation for Economic Co-operation and Development, 1967.

Klarner, P., and Raisch, S. "Move to the beat-Rhythms of change and firm performance," *Academy of Management Journal*) 2012, p amj. 2010.0767.

Koch, S. "Profiling an open source project ecology and its programmers," *Electronic Markets* (14:2) 2004, pp 77-88.

Krishnamurthy, S. "Cave or community?,") 2002.

Krishnamurthy, S. "On the intrinsic and extrinsic motivation of free/libre/open source (FLOSS) developers," *Knowledge, Technology & Policy* (18:4), 2006/12/01 2006, pp 17-39.

Leana, C.R., and Barry, B. "Stability and change as simultaneous experiences in organizational life," *Academy of Management Review* (25:4) 2000, pp 753-759.

Lee, S.Y.T., Kim, H.W., and Gupta, S. "Measuring open source software success," *Omega* (37:2) 2009, pp 426-438.

Lerner, J., and Tirole, J. "Some simple economics of open source," *The journal of industrial economics* (50:2) 2002, pp 197-234.

Levitt, B., and March, J.G. "Organizational learning," *Annual review of sociology*) 1988, pp 319-340.

Lindberg, A. "Understanding Change in Open Source Communities: A Co-evolutionary Framework," Academy of Management Proceedings, Academy of Management, 2013, p. 16619.

March, J.G. "Footnotes to organizational change," *Administrative science quarterly*) 1981, pp 563-577.

March, J.G. "Exploration and exploitation in organizational learning," *Organization science* (2:1) 1991, pp 71-87.

Méndez-Durón, R., and García, C.E. "Returns from social capital in open source software networks," *Journal of Evolutionary Economics* (19:2) 2009, pp 277-295.

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

Montgomery, D.C., and Woodall, W.H. "An overview of six sigma," *International Statistical Review* (76:3) 2008, pp 329-346.

Murugappan, M., and Keeni, G. "Blending CMM and Six Sigma to meet business goals," *Software, IEEE* (20:2) 2003, pp 42-48.

Nelson, R.R., and Winter, S.G. *An evolutionary theory of economic change* Harvard University Press, 2009.

Page, S.E. *Diversity and complexity* Princeton University Press, 2010.

Pentland, B.T., Hærem, T., and Hillison, D. "The (N) ever-changing world: stability and change in organizational routines," *Organization Science* (22:6) 2011, pp 1369-1383.

Rabiner, L.R. "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE* (77:2) 1989, pp 257-286.

Robbins, J.E., and Redmiles, D. "Software architecture design from the perspective of human cognitive needs," Proceedings of the California Software Symposium (CSS'96), 1996, pp. 16-27.

Roberts, J.A., Hann, I.-H., and Slaughter, S.A. "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.

Robinson, W.N., and Deng, T. "Data Mining Behavioral Transitions in Open Source Repositories " Hawaii International Conference on Software Systems, IEEE, HI, USA, 2014, p. (best paper nominee).

Robles, G., and González-Barahona, J.M. "A comprehensive study of software forks: Dates, reasons and outcomes," in: *Open Source Systems: Long-Term Sustainability*, Springer, 2012, pp. 1-14.

Salvato, C. "Capabilities unveiled: The role of ordinary activities in the evolution of product development processes," *Organization Science* (20:2) 2009, pp 384-409.

Santos, C., Kuk, G., Kon, F., and Pearson, J. "The attraction of contributors in free and open source software projects," *The Journal of Strategic Information Systems* (22:1) 2013, pp 26-45.

Scacchi, W. "Free and open source development practices in the game community," *Software, IEEE* (21:1) 2004, pp 59-66.

Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. "Understanding free/open source software development processes," *Software Process: Improvement and Practice* (11:2) 2006, pp 95-105.

Schroeder, R.G., Linderman, K., Liedtke, C., and Choo, A.S. "Six Sigma: Definition and underlying theory," *Journal of operations Management* (26:4) 2008, pp 536-554.

Sen, R., Singh, S.S., and Borle, S. "Open source software success: Measures and analysis," *Decision Support Systems* (52:2) 2012, pp 364-372.

Shannon, C.E. "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review* (5:1) 2001, pp 3-55.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. "Impacts of license choice and organizational sponsorship on success in open source software development projects," *Information systems research* (17:2) 2006, pp 126-144.

Subramaniam, C., Sen, R., and Nelson, M.L. "Determinants of open source software project success: A longitudinal study," *Decision Support Systems* (46:2) 2009, pp 576-585.

Subramanyam, R., and Krishnan, M.S. "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *Software Engineering, IEEE Transactions on* (29:4) 2003, pp 297-310.

Van de Ven, A.H. *Engaged Scholarship: A Guide for Organizational and Social Research: A Guide for Organizational and Social Research* Oxford University Press, 2007.

Von Krogh, G., Spaeth, S., and Lakhani, K.R. "Community, joining, and specialization in open source software innovation: a case study," *Research Policy* (32:7) 2003, pp 1217-1241.

Yourdon, E. *Death march* Pearson Education, 2003.

APPENDIX

Of the 18 Git repository activities, we focused on six, which most closely associated with development teamwork are in Table 5.

Table 5 GitHub development teamwork event types.

Event type	Description
IssuesEvent	An issue is created, closed, or reopened.
PushEvent	Code is committed (pushed) to the repository.
PullRequestEvent	A user requests that new code be pushed to the repository.
IssueCommentEvent	A comment is associated with an issue.
CommitCommentEvent	A comment is associated with a commit (PushEvent).
PullRequestReviewCommentEvent	A comment is associated with a PullRequest.

We obtained a total of 92,988 work motifs. The summary statistics of these motifs is provided in Table 6.

Table 6 Summary Statistics of Work Motifs

	Issue Event .*	PullRequest Event .*	Reopen Event .*
Number	68,095	21,776	3,117
Average Duration	53 days	17 days	46 days
Average number of Comments	2.68	2.27	3.83
Number of Unique Actors	2.7	2.5	2.6
Number of Events	5.4	5.7	7.2

Table 7 GitHub projects.

Project Name	Owner
AFNetworking	AFNetworking
android	github
android-bootstrap	AndroidBootstrap
Android-PullToRefresh	chrisbanes
AngularJS-Learning	jmcunningham
annotated_redis_source	huangz1990
async	caolan
atom	atom
AwesomeMenu	levey
backbone-boilerplate	backbone-boilerplate
bash-it	revans
bootstrap-sass	twbs

brackets	adobe
capistrano	capistrano
cocos2d-html5	cocos2d
coffeescript	jashkenas
colour-schemes	daylerees
compass	chriseppstein
coursera	coursera-dl
cw-omnibus	commonsguy
devise	plataformatec
discourse	discourse
docker	dotcloud
ember.js	emberjs
fabric.js	kangax
fastclick	ftlabs
FlatUIKit	Grouper
flight	flightjs
Font-Awesome	FortAwesome
Front-end-Developer-Interview-Questions	darcyclarke
Ghost	TryGhost
gitlabhq	gitlabhq
GMGridView	gmoledina
grunt	gruntjs
guzzle	guzzle
hackathon-starter	sahat
handlebars.js	wycats
highlight.js	isagalaev
history.js	browserstate
idiomatic.js	rwaldron
intro.js	usablica
jasmine	pivotal
javascript-patterns	shichuan
jekyll	jekyll
jqGrid	tonytomov
jQuery-menu-aim	kamens
jquery-pjax	defunkt
jScrollPane	vitch
KineticJS	ericdrowell
less.js	less
libgdx	libgdx
masonry	desandro
meteor	meteor

metrics	dropwizard
moment	moment
MWFeedParser	mwaterfall
netty	netty
NewsBlur	samuelclay
node-webkit	rogerwang
normalize.css	necolas
onepage-scroll	peachananr
OpenTLD	zk00006
parallax	wagerfield
phantomjs	ariya
phonegap-plugins	purplecabbage
platform_frameworks_base	android
Probabilistic-Programming-and-Bayesian-Methods-for-Hackers	CamDavidsonPilon
ProjectTox-Core	irungentoo
pure	yui
raphael	DmitryBaranovskiy
ratchet	twbs
ReactiveCocoa	ReactiveCocoa
resque	resque
retire	karmi
rubinius	rubinius
select2	ivaynberg
Semantic-UI	Semantic_Org
SlidingMenu	jfeinstein10
statsd	etsy
storm	nathanmarz
Telescope	TelescopeJS
TimelineJS	NUKnightLab
typeahead.js	twitter
underscore	jashkenas
Vundle.vim	gmarik
wysihtml5	xing
x-editable	vitalets
zepto	madrobby