Evolutionary Software Requirements Factors and their Effect on Open Source Project Attractiveness

Radu E. Vlas
University of Houston-Clear Lake
vlas@uhcl.edu

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

Cristina O. Vlas
University of Texas at Dallas
cristina.vlas@utdallas.edu

Abstract

Successful projects effectively manage their requirements. How the mix of different requirements evolves throughout a successful project life-cycle is poorly understood. Moreover, requirements practices may be changing, according to the authors of the New RE—a model of six critical requirements factors. The New RE focuses on leveraging existing components to create new functionality. This practice is also central to open-source development. Thus, to understand the proposed New RE model and its relationship to open-source development, in this study, we analyze over 200 projects from GitHub.com and compare them with a prior analysis of 31 projects from SourceForge. The results show that many of the proposed New RE factors are related to project attractiveness, which is important for open-source project success.

1. Introduction

The difficulty of requirements engineering (RE) tasks "has shifted from managing internal complexity to adapting and leveraging upon external and dynamic complexity."[1] Jarke and Lyytinen argue that software design is "is more about adjusting multiple interconnected software systems and components and improving their environmental "fit" by adapting them into a growing number of technical, social, and organizational subsystems." This growing design paradigm, that of reuse and adapting rather than designing from a blank slate, is not just a general RE concern, but is also an open-source development concern.

Recently, the established practice of modularization [2] has been elaborated to explain a form of emergent open-source coordination, called *superposition* [3]. This practice addresses requirements evolution by supporting design evolution through adaption. Superposition is the result of development behavior, in which (potentially dispersed) code is augmented to fulfill new functionalities (similar to aspect-oriented programming [4]). This theory asserts that developers aim to contribute independent work with few dependencies: "[t]hese changes layered on top of each other over time, each conceived and implemented for their own sake, yet simultaneously creating the circumstances taken as given for the production of the next layer in a way analogous to the superposition of rock strata."[3] This development technique supports independent, evolutionary software development.

Both the New RE and open-source superposition, assert that modern software development activities are focused on incremental, evolutionary design adaptation. We aim to measure software projects to understand if, and how, these two theories are instantiated in practice. In this study, we provide a means to measure New RE evolutionary practices. These measures and then correlated with project attractiveness, which is important for open-source project success.

1.1 The New RE

According to Jarke *et. al.*, requirements engineering (RE) is changing. "Despite its success over the last 30 years, the field of Requirements Engineering is still experiencing fundamental problems that indicate a need for a change of focus to better ground its research on issues underpinning current practices" [5]. We posit that these practices have changed significantly in recent years. We identify four *new* principles that underlie contemporary requirements processes, namely: (1) intertwining of requirements with implementation and organizational contexts, (2) dynamic evolution of requirements, (3) emergence of architectures as a critical stabilizing force, and (4) need to recognize unprecedented levels of design complexity." [5] Their paper summarizes changing research and practices in support of their assertion. Finally, they present potential new practices, for each of the four new principles. Within the second principle, named *evolve designs and ecologies*, they present four potential new practices in a form similar to CMM practices [6]:

- SG 2 Manage Requirements in Context
- SP 2.1 Monitor and evolve customer requirements
- SP 2.1 Monitor and evolve context requirements
- SP 2.1 Monitor product satisfaction of requirements (continuous validation)

These practices focus on monitoring requirements, mainly in support of managing their continuous change—a theme intertwined throughout the four new principles. In theory, awareness of the changing requirements will aid their management, which in turn will improve software development. The particulars of what requirements

URI: http://hdl.handle.net/10125/41806
ISBN: 978-0-9981331-0-2

qualities should be monitored is addressed in recent editorial from [1], which we consider next.

1.2 Six V's of The New RE

Classically, requirements engineering has focused on consistency, correctness, and completeness of the requirements document [7-9]. From the perspective of the New RE, addressing issues of requirements within a complex environment is central: "Whereas most of the interest in the past focused on understanding and managing the inner and static complexity of the design task by using abstraction, modularization, and related principles, today's complexity is of a different ilk. It is also external and dynamic." Their Six-V requirements measures illustrate how to address RE qualities in the New RE world.

Many of the V-measures are long-held qualities in requirements engineering, which have simply been renamed for alliteration. These include the first three V's of **Error! Reference source not found.**. The last three V's are presented as new measures, although some RE researchers may take issue with the novelty characterization—certainly, vagueness and variance have been concerns, and in fact are supported by research and tools [10-13]. Most, however, would agree with the general view presented: RE needs modern measures for the New RE, especially regarding measures of external and dynamic complexity.

The Six-V model is a modern interpretation of established measures. This study takes the model as given, rather than justify or extend the theory. Herein, we simply aim to assess the value of this model. The results may then be used to justify or extend the proposed Six-V model.

Consider modern agile development, where the project dashboard is critical to managing projects [14]. The centerpiece of these dashboards are burndown charts, which graph progress toward work completion [15]. Based on characteristics (e.g., slope, x-intercept) of such charts, managers can recognize and recover from potential project failure. For the New RE, one can envision dashboards graphically displaying assessments of the Six-V's, thereby providing a modern assessment through requirements. This is critical because managing requirements is often cited as the most important factor in determining project success [16, 17].

1.3 Open Source Requirements Engineering

Many open source projects are successful [22, 23]. In open source, the software product is developed, distributed, and supported by users. Common characteristics are (1) many developers, (2) volunteering rather than delegating, (3) limited emphasis on design activities, and (4) few plans, list of deliverables, or timelines[24]. Requirements are not represented in a classic requirements documents.

Table 1. Six "V's" of Requirements [1].

Feature	Definition	Classic RE	New RE
Volume	The size of the requirements pool influencing the scope of the work	Major focus of RE as influences effort estimation Medium to Large	Significant during RE as influences effort estimation [18] Large to Ultra-large
Veracity	To what extent requirements express the needs of the stakeholders and are consistent	Emphasized as the key feature of RE task, works well if requirements can be frozen	Important as an ideal but not key feature of most RE efforts [5, 19]
Volatility	The rate at which the requirements change over a given period of time	Recognized as a key reason for the failure of waterfall, e.g. [20]	Constant feature of software development for most environments [18]
Vagueness	To what extent designers and other stakeholders understand the content and consequences of the requirement	Not recognized as an important element other than to be avoided during RE task	Inherent feature of many RE initiatives due to initial lack of user learning or understanding of the dynamism introduced by the software in the environment
Variance	The variation in the design scope and consequences of the requirement pool and the heterogeneity of design components involved	Not recognized as an important element in RE activity	Significant element influencing RE dynamics and complexity. [5, 18]
Velocity	The rate at which requirements are changing over time	Not important and recognized	Significant contributor at specific context of RE especially in software platforms [21]

In open source development, many developers are also product users. They are stakeholders expressing needs that define system requirements [25]. It may appear that the requirements analysis stage is absent. However, Scacchi has identified software informalisms, which are "the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in an open source project" [26]. Scacchi identifies two dozen types of software informalisms, which include chats, email, forums, project digests, etc. By analyzing these unstructured, informal, natural language artifacts, one can better understand the requirements, and thus open source development. Such requirements analysis may help to predict successful projects.

One can apply text-mining techniques to classify software informalisms as kinds of requirements [27-32]. In the case of a SourceForge project, one can apply text-mining techniques to interpret the feature requests as requirements and their associated qualities. This provides a mechanism for analyzing the Six-Vs, both for research as well as presenting a modern requirements dashboard.

1.4 Project Attractiveness

Open source projects need to attract users and developers to keep a project active and successful [33-36]. Important success factors include, developer motivation and interest [37-42], and user interest [43]. Projects also have a self-reinforcing effect of attractiveness [44]. Users, often serving as the observing "eye balls" to bugs [45], contribute to a project's success. Hence, it is important for an open source project to attract both developers and users. Scweik et al. showed that for each developer added to an open source project, the chances of success increases 1.24 times [46]. Several studies attempted to identify what makes an open source project favored by developers and users. Drivers of attractiveness include contributors' intrinsic and extrinsic motivations for joining open source projects [38-42], contextual factors of the project [44], visibility of the project, and the work activities performed towards software maintenance and improvement [44]. To understand better the requirements context, we will analyze the relationship between requirements measures and project attractiveness.

1.5 Sustained Participation

An open source project cannot survive without sustained participation. Success, which has been extensively examined in the open source literature, is mostly measured at one time. Sustained participation, on the other hand, focuses on long established open source projects. Considering that 80 percent of open source projects fail, not due to quality, but because of insufficient long-term participation [47], it is important to predict sustained participation. Fang and Neufeld [42] investigate why developers continually contribute to open source projects in a sustainable way. Results show that situated learning and identity construction behaviors are associated with sustained participation. Qureshi and Fang [48] examine growth patterns of developers' socialization behavior and how that relates to their status progression. They identify four groups of newcomer behavior, based on the initial level of social resources of the developer and the growth rate of his/her socialization. The software development platform contributes to the socialization process.

GitHub.com is an example of a social-coding development-platform [49, 50], which supports rich, developer communications. Dabbish, et al. [51] found that developers use social coding capabilities for complex social activities, such as "inferring someone else's technical goals and vision when they edit code, or guessing which of several similar projects has the best chance of thriving in the long term. Users combine these inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation." Thus, people that are attracted to successful projects will follow them or download their code. Measures for tracking sustained participation include the number of developers and users and their various contributions over time, as well monitoring the projects that they follow.

1.6 Measuring Project Attractiveness

There are a number of ways to measure open-source project attractiveness. Two ways are stars and forks. When a project is starred, it is a kind of web bookmark, allowing a person to follow the project's activities. A fork is a kind of project copy, more common to developers who want to review or contribute to the code base. Both of these measures allow one to monitor the attractiveness of a project. We use these measures in our analysis of GitHub projects.

1.7 Article Overview

In this article, we present our study of how the Six-V requirements model relates to project attractiveness. Previously, Vlas and Robinson analyzed 31 projects from SourceForge, in a similar study [52]. Here, we develop a slightly different six-V measurement model and analyze the correlation between the Six-V's and project attractiveness. Herein, we study 248 projects from GitHub, where two of the Six-V metrics are new. We set out to confirm the findings of the prior analysis with a larger data set from a different repository. (Note that many GitHub projects are scripting projects, compared to the standard programming projects from SourceForge.) Our results here confirm the prior study, but with higher statistical significance. There are also other significant differences, which we elaborate in later sections. In short, the Six-V model helps monitor requirements and relate their qualities to project attractiveness. Next, we introduce the research hypotheses, followed by the research design, results, and finally conclusions.

2. Research Hypotheses

Having introduced related research on project attractiveness and the Six-Vs of requirements engineering, we now present our research model, consisting of six hypotheses.

2.1 Hypotheses

Following Jarke and Lyytinen, we start with the volume of requirements, defining it as "the size of the requirements pool influencing the scope of the work." [1] We adopt the generally accepted assumption that requirements reflect stakeholders' needs. Therefore, a large volume of requirements may have a positive effect—it indicates a large volume of needs and, in the context of open source, a large interest in the software artifact under development. A larger interest in an open-source project leads to a larger pool of contributors and a larger volume of discussions describing the needs and preferences of the project community. This helps improve the overall quality of the software artifact, and consequently, its attractiveness and success. A large volume of discussions may also have a negative effect—indicating either: (a) a lack of consensus among community participants, or (b) an inability of the developers to convert community needs and preferences into software artifact features. Given these two perspectives, positive and negative, on the volume of the requirements discussions, we interpret the volume of requirements as having an inverse U-shaped relationship with project attractiveness. According to our interpretation, at lower values of requirements volume, increases in volume have a positive effect on project attractiveness (via increased interest). At higher values of requirements volume, further increases in volume have a negative effect on project attractiveness (via increased dissonance).

Hypothesis 1 volume: Requirements volume has a curvilinear effect on open-source project attractiveness.

Jarke and Lyytinen define requirements velocity as the rate at which project requirements change over time. We apply this perspective to open-source development. In GitHub.com, the initial assertion of a requirement is established with the posting of an issue. The subsequent comments to that issue (i.e., the threaded conversation) are the changes, until the requirement/issue is closed. Our velocity metric counts the number of events, from issue open, through modifications, to issue close. High velocity means many steps that a requirement goes through before its closing. We interpret this as requirements dissonance and a sign of instability within the project. Consequently, we expect high velocity to have a negative effect on a project community's perceptions of project attractiveness.

Hypothesis 2 velocity: Requirements velocity has a negative effect on open-source project attractiveness.

Requirements volatility is defined as a rate of change of requirements content—meaning the topics of discussion[1]. Such volatility is inevitable, as it is arises from the innate variance within the pool of features that can fulfill project goals. Requirements volatility indicates a discussion of the goals or the means to fulfill those goals. However, after a threshold, increased volatility suggests a lack of focus, and the inability to respond consistently to stakeholders' needs. Therefore, we claim that volatility has a negative effect on project attractiveness.

Hypothesis 3 volatility: Requirements volatility has a negative effect on open-source project attractiveness.

Requirements vagueness is the extent to which requirements exhibit ambiguity. Requirements ambiguity impedes developers' ability to understand the needs and preferences of stakeholders. It impedes the ability of an open-source community to focus efficiently on topics of interest and value to the project, or to work efficiently towards specifying consistent requirements. Consequently, a higher value of vagueness is associated with an increased likelihood of wrong assumptions and interpretations, leading to a bad project with reduced attractiveness.

Hypothesis 4 vagueness: Requirements vagueness has a negative effect on open-source project attractiveness.

Table 2. Variable Operationalizations and Hypothesized Influence on Attractiveness.

Variable	Interpretation	Operationalization	H
Volume	Amount of project requirements	Count of requirements per data window	∩
Veracity	The consistency and fidelity of the requirements in expressing stakeholder needs	Count requirements within categories of completeness, consistency, and accuracy per data window	+
Volatility	Rate of change in the focus on a key subset of requirements over time	Total change in requirements category rankings, as calculated between adjacent data windows; the more requirements in a category, the higher the ranking.	-
Vagueness	Amount of ambiguity present in requirements	The inverse of the count of requirements categorized as unambiguous	-
Variance	Rate of change in the concepts represented in requirements over time	Count of requirements types that appear or disappear between adjacent data windows	+
Velocity	The rate at which the requirements are changed	The rate of change in the average workflow length per data window	-

A fifth factor described by Jarke and Lyytinen as defining the new requirements engineering is veracity. Requirements veracity is the extent to which requirements are consistent and express the needs of the stakeholders [1]. We interpret requirements veracity as a measure of the extent to which requirements (a) express consistent points of view, (b) comprehensively express the needs of stakeholders, and (c) are accurate. A high value of veracity indicates a good match between requirements and stakeholders' needs. This has a positive effect on the perceived attractiveness of the software artifact.

Hypothesis 5 veracity: Requirements veracity has a positive effect on open source project attractiveness.

Requirements variance is defined a measure of design-related variability and heterogeneity[1]. We measure this as the changes in the mix of requirements types at various periods within a project. A high value for variance indicates that many requirement types are considered. We interpret no-longer-considered requirements types as describing features that have been implemented within the software artifact, and newly-considered requirements types as new directions for the project. Both cases are indications of progress. Therefore, we conclude that variance has a positive effect on the attractiveness of the project.

Hypothesis 6 variance: Requirements variance has a positive effect on open source project attractiveness.

3. Research Design

3.1 Data Selection

We collect data from 272 open-source projects from GitHub. We did not constrain data collection to any specific time frames. To obtain a sample with variation among successful projects, we use 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 approximately 68 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 (as found in SourceForge). The initial projects were reduced to the final 248 due to data issues.

3.2 Data Preparation

KNIME workflows automate our data acquisition and preparation. Data was obtained directly from GitHub.com and stored into a SQL database. The GitHub data is comprised of 16 collections, which are combined, through filtering and joining, into a single table for data mining. Our data was derived from these collections: issues, issue events, issues comments, pull requests, and pull request comments. Each record in the table provides a vector for input into our data mining process.

The table represents a sequence of Git events. Of the 18 Git events, we focus on six, which most closely associate with software development:

1. IssuesEvent: An issue is created, closed, or reopened.
2. PushEvent: Commit (push) code to the repository.
3. PullRequestEvent: A user requests that new code be pushed to the repository.
4. IssueCommentEvent: Comment associated with an issue.
5. CommitCommentEvent: Comment associated with a commit (PushEvent).
6. PullRequestReviewCommentEvent: A comment is associated with a PullRequest.

From these events, we obtain text, which we analyze for requirements. Additionally, we characterize workflows to place the requirements in context. For example, these workflows allow us to characterize the number of events associated with requirements, which we use to characterize requirements velocity.

3.3 Development Workflows as Motifs

Git events, such as push and commit, represent work; however, the context of the work is missing. Work in most GitHub projects begins with an IssueEvent or a PullRequestEvent. Both represent a typical unit of development work, which may be scheduled, opened, closed, reopened, etc. Each contains text of requirements that guide software development. An IssueEvent typically represents a bug or enhancement. It follows a lifecycle of being opened, followed by code changes represented by commits, and then an issue close. For example:

IssuesEvent.open, PushEvent, PushEvent, IssuesEvent.close

Other events may intervene (e.g., comment events), as well as the issue may be reopened or never closed.

The PullRequestEvent is similar to the IssueEvent, but the subsequent work events are related to integrating the new code into the project's code repository.

A rule-based system is applied to recognize event sequences beginning with IssueEvent or a PullRequestEvent. We think about them as design workflows, which are initiated in response to a work request (e.g., issue or pull request). However, we use the more neutral term, motif, to indicate recognition of these common sequence patterns.

The rule-based system recognizes two kinds of work motifs in Git events. The basic form is as follows:

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

As indicated above, a work motif begins with either an IssueEvent or PullRequestEvent, followed by any other Git event that references the initiating event (by number). The motif records the initial event, and all subsequent events (and their attributes). When either an IssueEvent or PullRequestEvent is reopened, it is consider a new instance of the second motif pattern (above). Thus, open and reopen are each considered the beginning of a work motif.

We use work motifs to characterize requirements velocity. The motif length is the number of Git events it contains. We calculate velocity as (MotifLength_w / MotifLength_{w-1}), where *w* represents a data window.

3.4 Data Windows

In support of trend analysis, we divide the timestamped project data into windows by date. Within each window, various measures are computed, and then compared between adjacent windows. Data mining with this approach is known as stream-mining [53]; panel data statistics are applicable to such windows [54]. Data window size can affect the analysis. After various tests to ensure sufficient data in each window, we settled on 4-week windows, which is also meaningful to development cycles of GitHub projects.

3.5 Recognizing Requirements

Text in various Git events is parsed and analyzed for the discovery of requirements and of requirements types. Here we use an adapted version of Vlas and Robinson's method of identifying and classifying requirements [28]. This method generates classifications for the identified requirements from a set of 23 defined requirements types [55].

3.6 Analysis Approach

The dataset is analyzed as panel data using STATA 13.0 tool. Subject to list-wise deletion, our final data set has 9,268 observations. Multiple observations for each project over time raises concerns of potential interdependence among observations, which is addressed by lagging all our predictor and control variables with one window, compared to our dependent variable. This procedure also supports the claimed causation between predictor and dependent variable. Our dependent variable is project attractiveness and is measured with the natural logarithmic function of number of forks. The hypothesized causation between the predictors and the dependent variable is modeled using linear panel regression. Poisson regression is used to test results' robustness. The Hausman test reveals that either random or fixed effects models are appropriate [51]. We choose the fixed effects model as it may better reflect the structure of our panel and the possible correlations that may exist within projects. To avoid an increase in multicollinearity, we start with a baseline model, which includes only controls, and sequentially add variables. We therefore build 8 models and compute the variance inflation factors (VIF) of the uncentered variables for each model [52]. The full model's VIF is 1.85, well below the recommended threshold of 10. Control and independent variables are standardized and lagged.

3.7 Dependent Variable

Our dependent variable is project attractiveness. We operationalize it with the natural logarithmic function of number of forks. Forks represent the interest of a user to use the project and are a proxy for the level of project attractiveness, because it reflects the popularity that each project has among users. The distribution of the original variable is highly skewed and therefore we log it. The resulting variable has a near normal distribution.

3.8 Independent Variables

We conceptualize a set of six predictor variables (volume, velocity, volatility, vagueness, veracity, and variance) as determinants of project attractiveness. In the following, we describe these six predictor variables and in the next section we report the regression results that test the relationships among the predictor and the dependent variables.

Volume. To operationalize the concept of requirements volume we count the total number of requirements within each data window. The identification of requirements within a data window is performed by using an adapted version of the requirements discovery process proposed by Vlas and Robinson [27].

Velocity. Vlas and Robinson previously operationalized velocity as the rate at which the volume of requirements changes over time [52]. This operationalization as an aggregate value at the data window level was justified by the infeasibility of a manual requirement-level data extraction (extremely time consuming and error-prone). In this study, we benefit from the availability of additional requirement-level data. We define velocity as the rate of change in the number of events within a requirement workflow (the sequence of events from the inception throughout the closing of the requirement). We interpret this as the velocity of an individual requirement, and it aligns with the traditional concept of requirements change.

Volatility. Following the definition of volatility as the rate of change in requirements content, we create a ranking of the requirements types present in a window based on the count of requirements within each type. We label the top-most rank in a data window as the focus of the data window. When there is a subset of two or more requirements types that have same number of requirements we rank them equally by assigning them the top-rank within the subset. We compute the volatility of an individual requirement type as the absolute value of the difference between its rank in current data window and its rank in previous data window. To compute the overall data window volatility we sum up all individual requirement type volatilities. This approach measures the extent to which requirements content type changes over time.

Vagueness. Open source requirements are present in software informalisms [25]. Capturing requirements vagueness requires the ability to identify ambiguity in textual data. This is highly dependent upon being able to capture and analyze the context of the item of interest, the requirement in our case. In text mining, capturing context is a major challenge. However, the identification of the inverse of vagueness (clarity) is not as dependent upon context. Thus, we first measure clarity by counting the number of requirements classified as simplicity, conciseness, or self-descriptiveness and we add them up. Second, we inverse the value of clarity and we interpret it as vagueness. This procedure allows us to measure a lack of clarity—in other words, vagueness.

Veracity. Veracity is defined as a measure of consistency and fidelity. Following this definition, we focus on requirements completeness, consistency, and accuracy. We interpret the count of all these requirements as a measure of consistency and of the match between users' needs and the features expressed by requirements.

Variance. To compute a measure of variability of a set of requirements, we first identify and count the requirements types present in current data window that were not present in previous data window. Second, we identify and count the requirements types not present in current data window but present in the previous data window. To compute the overall variance of a data window we sum up all identified requirement types.

3.9 Control Variables

We control for a number of project characteristics that may explain project attractiveness. *Project stars* reflects the popularity that each project receives. As projects receive stars from users, they may become more attractive and therefore may influence the number of forks each project receives. *Project age* reflects the time that has elapsed since the start of the project (in weeks). Because users' interest in the projects increase with time, project attractiveness may also be confounded by the passing of time. *Commits* represent updates made to the project. Committed updates are likely to affect the attractiveness of the project by raising users' awareness of project quality. *Total event size* represents the total number of Git events in workflows and reflects changes made to the project or how active it is. *Total event duration* represents the time length of a work motif or how long it takes for an IssueEvent or PullRequestEvent to be closed. *Comments* represent the total count of comments associated with an issue, PullEvent or PullRequest in the workflow. These variables affect the complexity of a project and how quickly an issue can be resolved. *LOC added* represent the number of lines of codes written and *LOC deleted* represent the number of lines of code deleted. Together, these variables can affect the complexity of the project and the difficulty of solving an issue related to the project. We control for time series with *Window fixed effects*.

4. Research Results

4.1 GitHub

We calculate descriptive statistics and correlations between variables using STATA. Project attractiveness has the highest correlation with the volume of requirements ($r = 0.36^*$), meaning that as the volume of requirements increases, project popularity also increases. The Appendix presents the results of linear panel fixed-effects regression. We start with a baseline model with control variables only. Models 1 and 2 test Hypothesis 1 suggesting that requirements volume has a curvilinear (inverse U shape) effect on projects' attractiveness. For this hypothesis to be supported, Model 1 must report a positive coefficient for the volume term at the first power and Model 2 must report a negative coefficient for the volume term at the second power while maintaining a significant effect for the first power term. All these conditions are met. Accordingly, we safely claim that Hypothesis 1 is supported.

Hypothesis 2 claims that requirements velocity negatively affects project attractiveness. In Model 3, the velocity coefficient is $\beta = -0.018$ significant at $p < 0.05$. As a result, Hypothesis 2 is supported.

Hypothesis 3 claims that requirements volatility has a negative effect on project attractiveness. The negative and significant coefficient for the volatility term ($\beta = -0.117$, $p < 0.001$) in Model 4 supports Hypothesis 3.

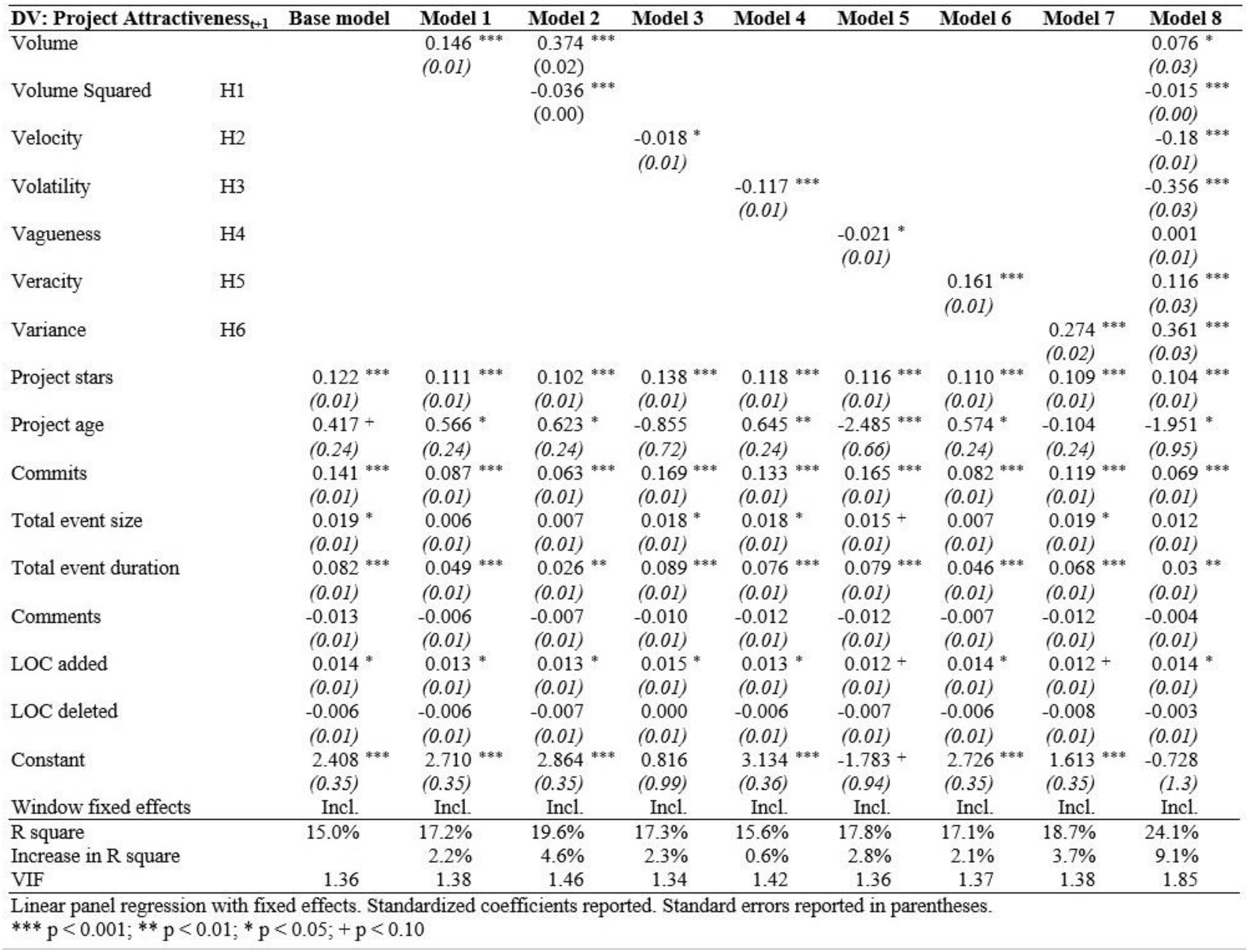
Hypothesis 4 claims that project attractiveness is negatively affected by requirements' vagueness. The negative and significant coefficient obtained in Model 5, $\beta = -0.021$ with $p < 0.05$, supports Hypothesis 4.

Hypothesis 5 claims that requirements veracity has a positive effect on project attractiveness. In Model 6, the coefficient for veracity is positive ($\beta = 0.161$) and significant at $p < 0.001$. Thus, Hypothesis 5 is supported.

Hypothesis 6 claims that requirements variance has a positive effect on project attractiveness. In Model 7, the coefficient for the variance term is positive ($\beta = 0.274$) and significant at $p < 0.001$. This result supports Hypothesis 6.

Model 8 is the full model. This model includes all six predictor variables. We find that, with the exception of vagueness, the effects of all predictors are significant and consistent with the hypothesized direction.

Table 3. Regression Results



DV: Project Attractiveness$_{t+1}$		Base model	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
Volume			0.146 ***	0.374 ***						0.076 *
			(0.01)	(0.02)						*(0.03)*
Volume Squared	H1			-0.036 ***						-0.015 ***
				(0.00)						*(0.00)*
Velocity	H2				-0.018 *					-0.18 ***
					(0.01)					*(0.01)*
Volatility	H3					-0.117 ***				-0.356 ***
						(0.01)				*(0.03)*
Vagueness	H4						-0.021 *			0.001
							(0.01)			*(0.01)*
Veracity	H5							0.161 ***		0.116 ***
								(0.01)		*(0.03)*
Variance	H6								0.274 ***	0.361 ***
									(0.02)	*(0.03)*
Project stars		0.122 ***	0.111 ***	0.102 ***	0.138 ***	0.118 ***	0.116 ***	0.110 ***	0.109 ***	0.104 ***
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
Project age		0.417 +	0.566 *	0.623 *	-0.855	0.645 **	-2.485 ***	0.574 *	-0.104	-1.951 *
		(0.24)	*(0.24)*	*(0.24)*	*(0.72)*	*(0.24)*	*(0.66)*	*(0.24)*	*(0.24)*	*(0.95)*
Commits		0.141 ***	0.087 ***	0.063 ***	0.169 ***	0.133 ***	0.165 ***	0.082 ***	0.119 ***	0.069 ***
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
Total event size		0.019 *	0.006	0.007	0.018 *	0.018 *	0.015 +	0.007	0.019 *	0.012
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
Total event duration		0.082 ***	0.049 ***	0.026 **	0.089 ***	0.076 ***	0.079 ***	0.046 ***	0.068 ***	0.03 **
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
Comments		-0.013	-0.006	-0.007	-0.010	-0.012	-0.012	-0.007	-0.012	-0.004
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
LOC added		0.014 *	0.013 *	0.013 *	0.015 *	0.013 *	0.012 +	0.014 *	0.012 +	0.014 *
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
LOC deleted		-0.006	-0.006	-0.007	0.000	-0.006	-0.007	-0.006	-0.008	-0.003
		(0.01)	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*	*(0.01)*
Constant		2.408 ***	2.710 ***	2.864 ***	0.816	3.134 ***	-1.783 +	2.726 ***	1.613 ***	-0.728
		(0.35)	*(0.35)*	*(0.35)*	*(0.99)*	*(0.36)*	*(0.94)*	*(0.35)*	*(0.35)*	*(1.3)*
Window fixed effects		Incl.	Incl.	Incl.	Incl.	Incl.	Incl.	Incl.	Incl.	Incl.
R square		15.0%	17.2%	19.6%	17.3%	15.6%	17.8%	17.1%	18.7%	24.1%
Increase in R square			2.2%	4.6%	2.3%	0.6%	2.8%	2.1%	3.7%	9.1%
VIF		1.36	1.38	1.46	1.34	1.42	1.36	1.37	1.38	1.85

Linear panel regression with fixed effects. Standardized coefficients reported. Standard errors reported in parentheses.
*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$; + $p < 0.10$

4.2 SourceForge Analysis and Comparison

The reported results of GitHub projects are robust and consistent with previous analysis reported by Vlas and Robinson on SourceForge projects [52]. Their analysis of 31 SourceForge projects over a 24 six-month long windows found that the attractiveness of open source projects (operationalized as download rate) is affected in a similar manner by the Six-V measures. Volume was found to display an inverse U-shape relationship with attractiveness, such that a high volume of requirements positively affected the download rate up to a threshold after which it had a negative effect. Velocity and volatility were hypothesized to negatively affect the download rate and support was found for the volatility-attractiveness relationship. Vagueness was conceptualized as “the extent to which designers make efforts to understand requirements” and its effect was found significant. Veracity and variance were hypothesized to positively affect the project attractiveness and support was found for veracity but not for variance.

We claim that our analysis brings further support for Jarke and Lyytinen’s [1] Six-Vs model, and it proposes improved operationalizations of these factors. While building on Vlas and Robinson [52], a comparison reveals significant changes. First, the dependent variable differs. While Vlas and Robinson [52] capture attractiveness with the number of downloads, in this study we operationalize it with the number of forks. This different operationalization of the same construct (attractiveness) enhances our understanding and builds robustness.

Second, herein we conceptualize velocity as the rate of change in the number of events within a workflow. This metric supports a correlation between velocity and project attractiveness. The prior study did not find support for this correlation [52]. Its velocity metric was an aggregate at the data window level, while our measure of velocity is at the requirement level, and thus, better aligned with the definition of the concept.

Third, in Vlas and Robinson [52], vagueness was the total number of requirements classified as relating to simplicity, conciseness, and self-descriptiveness. These three categories were used under the assumption that requirements in these categories suggest an existing necessity to fix problems of clarity. This approach to vagueness indirectly depicts vagueness as the need for clarity. Therefore, it was hypothesized to have a positive effect on attractiveness. Here, we take a more direct and intuitive approach and operationalize vagueness as the inverse of clarity. To capture clarity we use simplicity, conciseness, self-descriptiveness, and a fourth category—communicativeness. Thus, we hypothesize a negative relationship to project attractiveness. The new measure is more exhaustive due to the inclusion of this fourth category. Moreover, our approach on vagueness better matches the original definition by Jarke and Lyytinen [1] as "the extent [to which] designers and other stakeholders understand the content and consequences of the requirement."

Fourth, we find support for the positive relationship between variance and project attractiveness. Our improved model over the control model results in a 3.7% increase in R square. This suggests a causation effect between topic variance in stakeholder discussions and project attractiveness.

5. Discussion

5.1 Robustness

We test the robustness of our analyses by running an additional regression test using STATA. Because our dependent variable is a count of forks (logged) and because fixed panel data models poorly estimate time invariant (or slowly changing) effects, which we may have in our dataset for some long-lifecycle projects, we consider a Poisson regression to test the robustness of our results. The results are mostly consistent with the results obtained from the fixed-effects panel data model. Volume, volatility, veracity and variance measures affect project attractiveness according to the hypothesized direction, which extends the explanatory power of our model. Velocity and vagueness were not found to be significant in the Poisson regression.

5.2 Contributions

The ability to compare results across open source project repositories is important. While comparing our GitHub results to those of Vlas and Robinson, who analyzed SourceForge projects, we identify a number of valuable contributions. First, our results strengthen the validity of perceiving the Six-Vs of requirements engineering as important and defining characteristics of requirements in modern, open-source projects.

Second, we find support for the effects of velocity and variance on project attractiveness, two hypotheses that were not supported in Vlas and Robinson. This may be attributed to the larger dataset in our study. We also claim that our operationalizations of the two factors are more accurate and better aligned with their corresponding definitions, as provided by Jarke and Lyytinen.

Third, we address better the challenge of effectively capturing the spirit of the requirement-level definitions of Six-V measures. While these requirement-level factors were previously measured in an aggregate form, we find operationalizations that bring out the individual requirement characteristics into their calculation.

5.3 Critical Assessment and Future Research

While our study provides new insights on the importance of Six-V measures on project attractiveness, we recognize two important issues that can provide promising opportunities for future research. First, we use a 4-week rather than a 6-month data window size. This allows us to capture more refined trends in project lifecycles, but it can also be limited in capturing trends of slow-moving projects. Second, we use a number of aggregate-level measures. It would be ideal to collect data at the individual requirement level, but this may only be possible through manual (time consuming and error-prone) methods that would very significantly limit the sample size. Future studies may consider a different data collection technique. Third, we acknowledge the external validity limitations of our study as our findings may apply to the open source context only. We identify future research avenues in the refinement of our text mining tools for a better identification of requirements. Finally, there are opportunities to extend our research to other areas of development and to an extended set of factors that might enhance understanding of the determinants of project success.

6. Conclusions

In the open source literature, success models are of great interest. While success has been mostly analyzed as a static concept, we posit and confirm that open-source success depends on the continuous developing of requirements. By building on a previous study, we refine the New RE model as related to project success and apply it to an extended dataset of open source projects. Our study provides more precise metrics and confirms the value of the Six-V model. Researchers and practitioners may find value in applying the Six-V model to understand how requirements development contributes to project success over time. This dynamic model, directly linking development activities to project success, appears to be significant but remains largely unexplored.

7. References

[1] M. Jarke and K. Lyytinen, "Editorial:“Complexity of Systems Evolution: Requirements Engineering Perspective”," *ACM Transactions on Management Information Systems (TMIS),* vol. 5, p. 11, 2015.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM,* vol. 15, pp. 1053-1058, 1972.

[3] J. Howison and K. Crowston, "Collaboration through open superposition: A theory of the open source way," *Mis Quarterly,* vol. 38, pp. 29-50, 2014.

[4] R. Filman, T. Elrad, and S. Clarke, *Aspect-oriented software development*: Addison-Wesley Professional, 2004.

[5] M. Jarke*, et al.*, "The brave new world of design requirements," *Information Systems,* vol. 36, pp. 992-1008 (most downloaded, 2011), 2011.

[6] C. M. U. Software Engineering Institute, "Capability Maturity Model Integration (CMMISM), Version 1.1," CMU/SEI-2002-TR-011, Software Engineering Institute, Pittsburgh, 20022002.

[7] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," *Information and Software Technology,* vol. 45, pp. 993-1009, 2003.

[8] A. Davis, Software Requirements: Objects, functions, and states: Prentice Hall, 1993.

[9] W. N. Robinson, S. Pawlowski, and V. Volkov, "Requirements Interaction Management," *ACM Computing Surveys (CSUR),* vol. 35, pp. 132 - 190, June 2003.

[10] J. A. Goguen, "Formality and Informality in Requirements Engineering," in *ICRE*, 1996, pp. 102-108.

[11] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Applications of linguistic techniques for use case analysis," *Requirements Engineering,* vol. 8, pp. 161-170, 2003.

[12] V. Alves, N. Niu, C. Alves, and G. Valença, "Requirements engineering for software product lines: A systematic literature review," *Information and Software Technology,* vol. 52, pp. 806-820, 2010.

[13] K. Pohl, Requirements engineering: fundamentals, principles, and techniques: Springer Publishing Company, Incorporated, 2010.

[14] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying SCRUM principles to software product management," *Information and software technology,* vol. 53, pp. 58-70, 2011.

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

[16] G. Standish. (1995). *CHAOS*. Available: www.standishgroup.com

[17] P. L. Bannerman, "Risk and risk management in software projects: A reassessment," *Journal of Systems and Software,* vol. 81, pp. 2118-2133, 2008.

[18] S. Hansen, N. Berente, and K. Lyytinen, "Requirements in the 21st century: Current practice and emerging trends," in *Design requirements engineering: A ten-year perspective*, ed: Springer, 2009, pp. 44-87.

[19] D. T. Ross, "Structured Analysis (SA): A language for communicating ideas," *Transactions on Software Engineering,* vol. SE-3, pp. 16-34, January 1977.

[20] J. Frederick P. Brooks, *The Mythical Man-Month*: Addison Wesley, 1995.

[21] M. Fisher, M. Abbott, and K. Lyytinen, The Power of Customer Misbehavior: Drive Growth and Innovation by Learning from Your Customers: Palgrave Macmillan, 2013.

[22] K. Crowston, J. Howison, and H. Annabi, "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice (Special Issue on Free/Open Source Software Processes.),* vol. 11, pp. 123-148, 2006.

[23] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code Quality Analysis in Open Source Software Development," *Information Systems Journal,* vol. 12, pp. 43-60, February 2002 2002.

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

[25] B. Fitzgerald, "The transformation of open source software," *MIS Quarterly,* vol. 30, pp. 587-598, 2006.

[26] W. Scacchi, "Understanding Requirements for Open Source Software," in *Design Requirements Engineering – A Multi-disciplinary perspective for the next decade*, K. Lyytinen*, et al.*, Eds., ed: Springer-Verlag, 2009.

[27] R. Vlas and W. N. Robinson, "Applying a Rule-Based Natural Language Classifier to Open Source Requirements: a Demonstration of Theory Exploration," in *Hawaii International Conference on Software Systems*, HI, USA, 2013.

[28] R. Vlas and W. N. Robinson, "A Pattern-Based Method for Requirements Discovery and Classification in Open-Source Software Development Projects " *Journal of Management Information Systems (JMIS),* 2012.

[29] R. Vlas and W. Robinson, "Extending and Applying a Rule-Based Natural Language Toolkit for Open Source Requirements Discovery and Classification " presented at the Open Source Systems (OSS'11), 2011.

[30] C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher, "Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes," in *International Requirements Engineering, 2008. RE'08. 16th IEEE*, 2008, pp. 165-168.

[31] J. Cleland-Huang, H. Dumitru, C. Duan, and C. Castro-Herrera, "Automated support for managing feature requests in open forums," *Communications of the ACM,* vol. 52, pp. 68-74, 2009.

[32] P. Laurent and J. Cleland-Huang, "Lessons learned from open source projects for facilitating online requirements processes," in *Requirements Engineering: Foundation for Software Quality*, ed: Springer, 2009, pp. 240-255.

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

[34] S. Koch, "Profiling an open source project ecology and its programmers," *Electronic Markets,* vol. 14, pp. 77-88, 2004.

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

[36] S. Krishnamurthy, "Cave or community?," 2002.

[37] A. Bonaccorsi and C. Rossi, "Why open source software can succeed," *Research policy,* vol. 32, pp. 1243-1258, 2003.

[38] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in Open Source projects: an Internet-based survey of

contributors to the Linux kernel," *Research policy,* vol. 32, pp. 1159-1177, 2003.

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

[40] J. A. Roberts, I.-H. Hann, and S. A. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Management science,* vol. 52, pp. 984-999, 2006.

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

[42] Y. Fang and D. Neufeld, "Understanding sustained participation in open source software projects," *Journal of Management Information Systems,* vol. 25, pp. 9-50, 2009.

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

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

[45] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy,* vol. 12, pp. 23-49, 1999.

[46] C. M. Schweik, R. C. English, M. Kitsing, and S. Haire, "Brooks' versus Linus' law: an empirical test of open source projects," in *Proceedings of the 2008 international conference on Digital government research*, 2008, pp. 423-424.

[47] J. Colazo and Y. Fang, "Impact of license choice on open source software development activity," *Journal of the American Society for Information Science and Technology,* vol. 60, pp. 997-1011, 2009.

[48] I. Qureshi and Y. Fang, "Socialization in open source software projects: A growth mixture modeling approach," *Organizational Research Methods,* vol. 14, pp. 208-238, 2011.

[49] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *Software, IEEE,* vol. 30, pp. 52-66, 2013.

[50] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 359-364.

[51] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, 2012, pp. 1277-1286.

[52] W. N. Robinson and R. Vlas, "Requirements Evolution and Project Success: An Analysis of SourceForge Projects," in *Association for Information Systems Conference (AMCIS)*, Puerto Rico, 2015.

[53] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *ACM Sigmod Record,* vol. 34, pp. 18-26, 2005.

[54] C. Hsiao, *Analysis of panel data*: Cambridge university press, 2014.

[55] J. A. McCall, P. K. Richards, and G. F. Walters, *Factors in Software Quality*: NTIS, 1977.

[56] J. Neter, W. Wasserman, and M. H. Kutner, *Applied linear statistical models* (2nd ed.) IL: Irwin Homewood, 1985.