Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments

March 2010

Manuel Sojer[1] and Joachim Henkel[1, 2]

[1]Technische Universität München, Schöller Chair in Technology and Innovation Management, Arcisstr. 21, D-80333 Munich, Germany.
sojer|henkel@wi.tum.de

[2]Center for Economic Policy Research (CEPR), London

Abstract

The focus of existing open source software (OSS) research has been on how and why individuals and firms add to the commons of public OSS code—that is, on the "giving" side of this open innovation process. In contrast, research on the corresponding "receiving" side of the innovation process is scarce. We address this gap, studying how existing OSS code is reused and serves as an input to further OSS development. Our findings are based on a survey with 686 responses from OSS developers. As the most interesting results, our multivariate analyses of developers' code reuse behavior point out that developers with larger personal networks within the OSS community and those who have experience in a greater number of OSS projects reuse more, presumably because both network size and a broad project experience facilitate local search for reusable artifacts. Moreover, we find that a development paradigm that calls for releasing an initial functioning version of the software early—as the "credible promise" in OSS—leads to increased reuse. Finally, we identify developers' interest to tackle difficult technical challenges as detrimental to efficient reuse-based innovation. Beyond OSS, we discuss the relevance of our findings for companies developing software and for the receiving side of open innovation processes in general.

Keywords: Innovation, software development, open source software, code reuse, software reuse

We are grateful to Oliver Alexy, Timo Fischer, Stefan Haefliger, Francesco Rullani, and seminar participants at the Pre-ECIS 2009 Open Source and Innovation Workshop, the TUM/Imperial Paper Development Workshop 2009, and the Open Source, Innovation, and Entrepreneurship Workshop 2010 for helpful comments.

1. Introduction

The public development of open source software (OSS)[1] is a specific instance of open innovation, a term coined by Chesbrough (2003). A large body of empirical work has addressed the "giving" side of this open innovation process, that is, exploring the question of why and how individuals (e.g. Ghosh et al., 2002; Hars and Ou, 2002; Hertel et al., 2003; Lakhani and Wolf, 2005; Henkel, 2009) and firms (e.g. West, 2003; Dahlander, 2005; Gruber and Henkel, 2005; Bonaccorsi et al., 2006; Henkel, 2006; Rossi Lamastra, 2009) make their developments freely available for others to use and build upon.

In contrast, research on the "receiving" side of the innovation process,[2] that is, on the extent, drivers, and impediments of *reuse* of existing OSS code in subsequent OSS development, is scarce and either based on high-level code or dependency analyses (German, 2007; Mockus, 2007; Spaeth et al., 2007; Chang and Mockus, 2008), or on case studies (von Krogh et al., 2005; Haefliger et al., 2008). While this research suggests that code reuse is of major importance for OSS development, a large-scale quantitative study of the phenomenon on the level of individual developers is lacking.

A better understanding of code reuse in OSS is desirable, not only in itself, but also because it will yield insights on reuse beyond OSS. Reuse has long been recognized as crucial to overcome the "software crisis" (Naur and Randell, 1968), as it allows for more efficient and more effective development of software of higher quality (Krueger, 1992; Kim and Stohr, 1998). More generally, the literature on innovation management points to knowledge reuse as an important factor mitigating the cost of innovation (e.g. Langlois, 1999; Majchrak et al., 2004). Despite significant advances in reuse research, especially software reuse in commercial firms is still not without issues and its antecedents are not fully understood yet (e.g. Desouza et al., 2006; Sherif et al., 2006). Some scholars suspect that reuse failure is often related to individual developer issues (e.g. Isoda, 1995; Morisio et al., 2002). However,

---

1 For better readability, we will use the term Open Source software in this article, but our work also refers to Libre and Free software, which differs from open source in ideological considerations but not in technical ones. See http://www.gnu.org/philosophy/free-sw.html for further information.

2 Also the users of OSS obviously receive code, however, since they do not base own innovations on it we do not consider them to be on the "receiving" side of the OSS innovation process.

there is a paucity of, especially quantitative, research addressing the view of individual developers on reuse (e.g. Sen, 1997; Ye and Fischer, 2005).

Our aim is to fill the above gap regarding the "receiving" side of OSS innovation and to leverage our findings to augment general software reuse literature by adding insights regarding the perspectives of individual developers on reuse with a survey-based empirical study of code reuse in public OSS development. We quantitatively assess the importance of code reuse as one form of reuse in software development in OSS, and explore its drivers and impediments at the level of individual developers. Our empirical approach relies on a web-based survey to which we had, via email, invited 7,500 developers from SourceForge.net, the largest OSS development platform.

Our results point out that code reuse does play a major role in OSS development; developers reported, on average, that 30 percent of the functionality they have implemented in their current main projects has been based on reused code. Investigating the drivers of reuse in multivariate analyses, we find that developers who believe in the effectiveness, efficiency, and quality benefits of reuse and developers who see reuse as a means to work on their preferred development tasks rely more on existing code. Further, presumably because both a larger network and experience in a greater number of projects provide them with access to local search for reusable artifacts, developers with larger personal networks within the OSS community and experience in a greater number of OSS projects reuse more. Moreover, we find that a development paradigm that calls for releasing an initial functioning version of the product early, and so delivering a "credible promise", leads to increased reuse. Finally, developers' interest to tackle difficult technical challenges is identified as detrimental to efficient reuse-based innovation, while developers' commitment to the OSS community leads to increased reuse behavior.

The remainder of the paper is organized as follows. The next section reviews relevant literature on software reuse and OSS, followed by a section that presents our research model and hypotheses. After that, we elaborate on our data and measures before we present our analyses and results. The last section concludes with a summary and a discussion. A supplemental appendix contains further tables referred to in the paper but not included in its main body for space considerations.

2. Literature Review

The theoretical foundation of this paper draws on two streams of the literature. First, we review relevant software engineering literature on reuse and its implementation in firms. Second, scholarly work on OSS development provides the context of our work, establishing basic concepts of why developers contribute to OSS projects and how they do so. A summary of the small base of scholarly work on code reuse in OSS development concludes the literature review.

2.1. Reuse in Software Development

Software reuse (as the software-specific form of knowledge reuse (e.g. Langlois, 1999; Majchrak et al., 2004)) is "[…] the process of creating software systems from existing software rather than building software systems from scratch" (Krueger, 1992, p. 131). The artifacts most commonly reused in software development are components (pieces of software that encapsulate functionality and have been developed specifically for the purpose of being reused) and snippets (multiple lines of code from existing systems) (Krueger, 1992; Kim and Stohr, 1998). Our study focuses on these two artifacts, and we refer to their reuse as "code reuse." Software reuse promises not only increased development efficiency and reduced development times, but also improved software quality and better maintainability because developers do not have to develop everything from scratch, but rather can rely on existing, proven, and thoroughly tested artifacts (Frakes and Kang, 2005).

Despite these compelling benefits, software reuse still fails frequently in commercial firms, sometimes for technical, but most often for human and organizational reasons (e.g. Morisio et al., 2002). The importance of the individual developer in successful reuse is undisputed. Isoda (1995, p. 183) for instance concedes, "unless they [software engineers] find their own benefits from applying software reuse […] they will not […] perform reuse." Still, there is a paucity in reuse research that focuses on the individual developer (Sen, 1997; Ye and Fischer, 2005).

OSS seems to be a unique opportunity to enhance our knowledge about the role of individuals in successful reuse-based innovation and software reuse in particular for two reasons. First, contrary to commercial software developers who

are often restricted to the limited amount of code available in their firms' reuse repositories, the abundance of OSS code available under licenses which generally permit reuse in other OSS projects provides OSS developers with broad options to reuse existing code if they wish to do so. Second, the broad scholarly knowledge about the motivations and beliefs of OSS developers should be helpful in analyzing the perspectives of individual developers on software reuse. The next section establishes community-based, public OSS development as the empirical setting of our analysis.

2.2. Open Source Software Development

Strictly speaking, software is OSS if it comes under an open source license. Such a license grants users of the software the right to access, inspect, and modify the source code of the software and distribute modified or unmodified versions of it.[3] Since much OSS is developed by informal collaboration in public OSS projects (Crowston and Scozzi, 2008), the term "OSS" is often also understood to imply that the software has been developed in the "OSS fashion" (von Krogh et al., 2008). Typically, the development of software in OSS projects differs strongly from the development of traditional software in most commercial setups (Crowston et al., 2009). In this context the motivation of developers to spend considerable time on their OSS projects and the process of OSS development are of particular relevance to our study.

A large body of literature has emerged that addresses the first topic. Common to most of this work is the finding that OSS developers work on their projects for both intrinsic and extrinsic reasons. As intrinsic motivations, scholars have identified identification with the OSS community and the resulting wish to support it (Hertel et al., 2003), ideological support of the OSS movement (Stewart and Gosain, 2006), the desire to help others (Hars and Ou, 2002), and, most importantly, the fun and enjoyment that developers experience when working on their projects (Lakhani and Wolf, 2005). Based on psychology research (Amabile et al., 1994), Sen et al. (2008) further differentiate fun into the enjoyment and "flow" feelings (Csíkszentmihályi, 1990) that developers perceive when writing code and the satisfaction of solving challenging technical problems. Extrinsic motivations of

[3] Whether a software license is an open source license is determined by the Open Source Initiative (http://www.opensource.org).

OSS developers may derive from the wish to enhance their reputation in the OSS community (Lakhani and Wolf, 2005), to hone their software development skills (Hars and Ou, 2002), to develop or adapt software functionality to their own needs (Hertel et al., 2003), and to signal their skills to potential employers and business partners (Lerner and Tirole, 2002). Also, they may be paid directly for their OSS work, for example, if it is part of their job (Ghosh et al., 2002).

Regarding the process of OSS development, OSS projects are often started by an individual developer who has a need for certain software functionality that does not yet exist (Raymond, 2001). After initialization, the developer typically wants to attract other developers to participate in the project. An incentive for others to join the project is that it offers interesting tasks and also seems feasible (von Krogh et al., 2003). The founder can enhance this recruitment process by delivering a "credible promise", which Lerner and Tirole (2002, p. 220) describe as "a critical mass of code to which the programming community can react. Enough work must be done to show that the project is doable and has merit." However, not only does the founder have to prove that the project is worthy of support by others, but also developers interested in joining a project often have to show that they possess the skills required by solving some of the technical issues the project is currently facing (von Krogh et al., 2003).

2.3. Code Reuse in Open Source Software Development

There is scant research on code reuse in OSS and so far no large-scale quantitative data on the developer level exist. Initial academic work, however, suggests that code reuse is practiced in OSS projects even at a high level. Analyzing the code of a large number of OSS projects, Mockus (2007) and Chang and Mockus (2008) measure the overlap of filenames among OSS projects in their database of 38.7 thousand OSS projects and conclude that about 50 percent of the components exist in more than one project. Mockus's (2007) data even suggests that code reuse is more popular in OSS development than in the traditional commercial closed source software arena. Following a different approach, both German (2007) and Spaeth et al. (2007) rely on dependency information available in Linux distributions to show that most packages in these distributions require other packages as they reuse their functionality.

Using case studies on the project and individual developer level rather than large-scale code analyses, von Krogh et al. (2005) and Haefliger et al. (2008) confirm that OSS developers reuse existing code—in the form of components and snippets—as well as abstract knowledge—such as algorithms and methods. Diving into the mechanics of code reuse in OSS, Haefliger et al. (2008) find that OSS developers reuse code because they want to make their development work more efficient, they lack the skills to implement certain functionality by themselves, they prefer some specific development work over other tasks. or they want to deliver a "credible promise" with their project. The authors further point out that there exist equivalents to some of the components of corporate reuse programs, such as the OSS repositories like SourceForge.net which can substitute internal reuse repositories within firms, or the reuse frequency of a component which can serve as a proxy for the component's its quality and thus substitutes certification.

3. Research Questions and Hypotheses

Building on the existing research on code reuse in OSS presented above, this paper seeks to use large-scale quantitative data obtained through a survey among OSS developers to answer the question: under what conditions do developers prefer reusing existing code over developing their own code from scratch? In this context, the following specific research questions will be addressed:

1. How important is code reuse in OSS development projects?
2. What do OSS developers perceive as the benefits of code reuse and what do they see as the issues and impediments?
3. How is the degree of code reuse in open source developers' work determined by their characteristics and those of their project?

The first question establishes if and to what extent OSS developers reuse existing code, while the subsequent questions explore how this behavior can be understood and explained. Question three will be addressed using regression analyses. To guide the choice of explanatory variables and formulate hypotheses, a research model is developed in the following section. To provide a solid theoretical base, our research model builds on the well-established Theory of Planned Behavior (TPB) (Ajzen, 1991) and is refined and extended with both interviews and literature on code reuse and OSS.

3.1. Theory of Planned Behavior

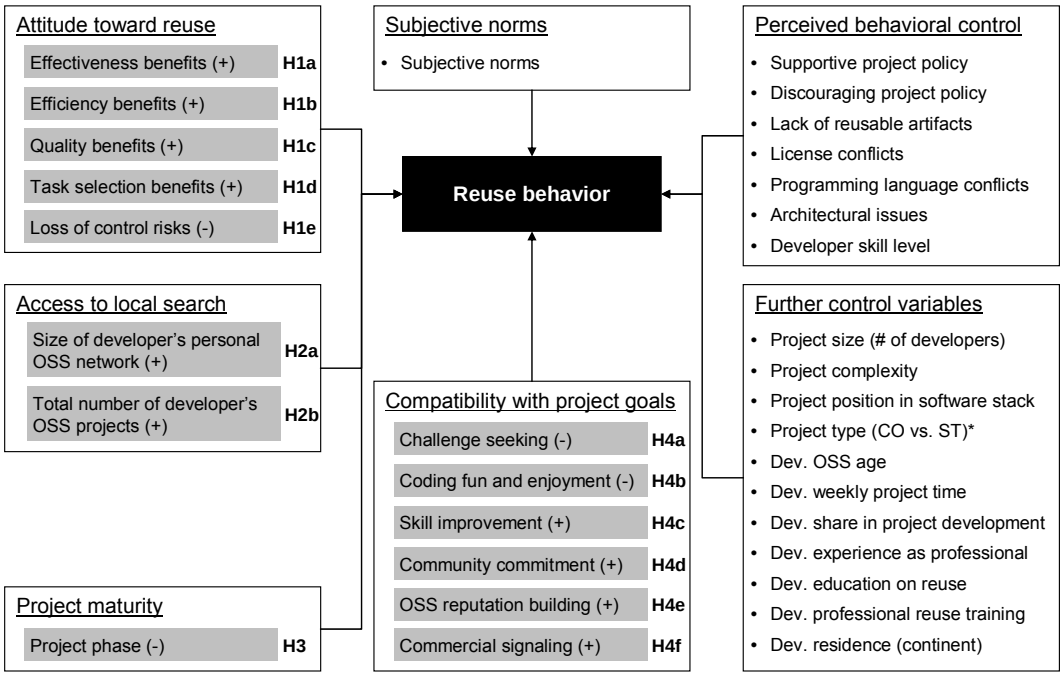
Initially developed in the context of social psychology, TPB as a behavioral model has found wide adoption in various fields of information systems (IS) research. TPB is a parsimonious and rather generic model explaining human behavior and thus provides an excellent starting point to investigate code reuse as one particular form of behavior. Research related to the topic of our study has relied on TPB or its sister model TAM (Technology Acceptance Model) (Davis et al., 1989) to explain for example software developers' application of various development methodologies such as CASE tools (Riemenschneider and Hardgrave, 2001), object-oriented software development (Hardgrave and Johnson, 2003) or generally formalized software development processes (Riemenschneider et al., 2002; Hardgrave et al., 2003). Following the encouraging results of this stream of research we base our research model on TPB.

TPB posits that behavior is determined by intention, which itself is predicted by three factors: (1) attitude toward the behavior, (2) subjective norms, and (3) perceived behavioral control. Attitude is formed by the individual's beliefs about the consequences and outcomes (both positive and negative) of the behavior. Subjective norms refer to pressure from the social environment as perceived by the individual to perform or not perform the behavior. Lastly, perceived behavioral control is the perception of individuals of their ability to perform the behavior. It can be further broken down in individuals' "capability" of performing the behavior and the "controllability" (Ajzen, 2002) the individuals have over the behavior, that is, whether the decision to perform the behavior is theirs or not.

3.2. Research Model and Hypotheses

Using TPB as a starting point for our research model (see Figure 1), we argue that developers' reuse behavior is influenced by their attitude toward code reuse, their subjective norms on code reuse, and the behavioral control they perceive regarding code reuse. Contrary to typical work relying on TPB, we do not employ generic scales to measure these constructs in most cases, but rather operationalize them with unique scales and single items explicitly framed in the OSS and code reuse context. As a second deviation from typical TPB research we will test the research model with different regressions which either use intention to reuse as the dependent variable or employ actual reuse behavior as the dependent

variable. Since we do not combine intention and behavior into one construct, but rather employ only one of them in each of our regression models we stay true to the TPB assumption that the two concepts are related but not the same. Comparing the results of the regressions with different dependent variables adds robustness to our findings.



Notes: The direction of the hypothesis is indicated by (+) and (-); *CO=component project, ST=standalone executable application project.

Figure 1: Research model

Note that our research model aims at explaining developers' reuse behavior without explicitly differentiating between component and snippet reuse. In conventional software development component reuse is typically considered as black-box reuse, implying that developers can neither access nor modify the source code of the components the reuse. Thus, component reuse is assumed to follow different drivers than white-box reuse (e.g. snippet reuse) where access to source code is given (Ravichandran and Rothenberger, 2003). In the context of OSS however, also the source code of components is available to reusing developers and our survey data indicate that about 50% of the developers exercise the option to modify it. Because of this we expect no fundamental differences in the drivers of

component and snippet reuse and treat both forms of code reuse jointly in our research model.

Based on our interviews[4] and existing research, we have identified five main drivers that influence developers' attitude toward code reuse since they determine whether developers expect positive or negative outcomes from reuse. These drivers are developers' perceptions of (1) the effectiveness of reuse, (2) the efficiency of reuse, (3) the software quality attained by reuse, (4) the task selection benefits resulting from reuse, and (5) the potential loss of control over their project that might come with reuse. The link between reuse and effectiveness, efficiency, and software quality is straightforward. In addition, code reuse might result in task selection benefits if developers can avoid certain tasks by reusing existing code (Haefliger et al., 2008). As the fifth driver, reuse can lead to control loss as a developer reusing code from another project might become dependent on this project to develop the code further, fix bugs, and so on. Since developers with a more positive perception of the above drivers should hold a more positive attitude toward reuse, TPB suggests that they rely more on reusing existing code in their work. Based on this logic, the following hypotheses can be derived for the five drivers:

Developers reuse more existing code…

H1a: …the more strongly they perceive the effectiveness benefits of reuse.

H1b: …the more strongly they perceive the efficiency benefits of reuse.

H1c: …the more strongly they perceive the quality benefits of reuse.

H1d: …the more strongly they perceive the task selection benefits of reuse.

H1e: …the less strongly they perceive the loss of control risks of code reuse.

Since the primary interest of our research is to understand how individual developer characteristics influence reuse, both subjective norms and perceived behavioral control as the two other parts of TPB besides attitude are treated as control variables in our model. The controllability portion of perceived behavioral

[4] See the next section for an overview of our interviews.

control is operationalized by six variables relating to project attributes. Two dummy variables indicate whether there exist policies in the project supporting or discouraging code reuse. Four Likert-scale variables capture the intensity of general impediments to code reuse: a lack of reusable code for the specific requirements of a developer's project; conflicts between the license of the developer's project and the license of the code to be reused; incompatibilities between programming languages, when the code to be reused is written in a different language than the developer's project (Haefliger et al., 2008), or when the programming language of the focal project makes it difficult to include code in foreign languages; and an architecture of the developer's project that is not modular enough to allow for easy reuse of existing code (Baldwin and Clark, 2006). The capability portion of perceived behavioral control is operationalized through each developer's self-reported skill level in software development, arguing that without some proficiency, developers will not be able to understand and integrate foreign code.

TPB research posits that attitude toward a behavior, subjective norms, and perceived behavioral control explain behavior comprehensively (Ajzen, 1991). We stay true to this assumption when we add further groups of hypotheses and control variables hereinafter because all of these additional groups could be incorporated into the three original TPB groups of attitude, subjective norms, and perceived behavioral control. We did, however, choose to display some hypotheses as independent groups to better illustrate the ideas behind them. Moreover, some further control variables are shown as a group of their own because their influence on attitude, subjective norms, and perceived behavioral control is rather indirect.

In the first additional hypotheses group, we argue that developers' access to local search leads to increased code reuse. Banker et al. (1993) show that developers will reuse if their costs for searching and integrating existing code are lower than for developing it from scratch. These costs for searching and integrating are lower if OSS developers can turn to their own experience or that of fellow OSS developers who can point them to the code they need, assure them of its quality, and explain to them how it works and how to best integrate it (Haefliger et al., 2008). Consequently, we posit that developers with a larger personal network of other OSS developers will reuse more code because they can reap the benefits of local search *(H2a)*. Similarly, developers who have been active in more OSS projects in the past

will also show increased code reuse behavior *(H2b)*. Summarizing, the following two hypotheses can be derived regarding developers' access to local search.

Developers reuse more existing code…

H2a: …the larger their personal OSS network.

H2b: …the greater the number of OSS projects they have been involved in.

Further, we also conjecture a relationship between the maturity of an OSS project and the code reuse behavior of its developers. As pointed out in the literature review section, OSS developers launching a project strive to deliver a "credible promise" as quickly as possible in order to attract other developers' support. Code reuse is an excellent tool to accomplish that because it allows the addition of large blocks of functionality to a new project with limited effort (Haefliger et al., 2008). Further, code reuse can help a new project to overcome its "liabilities of smallness" (Aldrich and Auster, 1986) and quickly close the gap to established competing projects in its domain. Lastly, while code reuse is very helpful in the early phases of the life of an OSS project, we expect its importance to decline once the project has reached a certain level of maturity. At that point, the project has implemented all required basic functionality and turns toward fine-tuning the aspects that make it unique, which by definition is difficult with reused code. Thus, we posit that the less mature an OSS project is, the more code its developers will reuse *(H3)*.

H3: Developers reuse more existing code the less mature their project.

In the final group of hypotheses, we argue that the compatibility of code reuse with developers' own goals in their project will influence the extent of their code reuse behavior. This is important because the "attitudes"-group of our model presented above captures developers' general attitude toward code reuse, while the "compatibility"-group presented in the following will help to link these general attitudes to the developers' work in one specific project. We follow Moore and Benbasat (1991, p. 195) and define compatibility as the degree to which code reuse "is perceived as being consistent with the existing values, needs, and past experiences" of an OSS developer and focus primarily on "values" and "needs"

("experiences" being addressed by *H2b*). Our argumentation regarding compatibility between developers' project goals and their reuse behavior is based on the motivations of developers to participate in OSS projects described earlier.

Sen et al. (2008) show empirically that developers for whom tackling difficult technical problems is a main motivation to work on their project try to limit the number of team members involved in their project besides themselves because they want to solve the problems themselves and without the help of others. In similar fashion, developers who work on their project to tackle difficult technical challenges should reuse less existing code because reuse would solve some of the challenges for them *(H4a)*. In order to be able to focus on solving these difficult technical challenges by themselves, developers might very well show increased reuse behavior for other parts of their project, but we control for this effect by including developers' perception of task selection benefits through reuse (see *H1d* above). Also supportive of our argumentation is DiBona et al.'s (1999, p. 13) description of the "satisfaction of the ultimate intellectual exercise" which developers feel "after completing or debugging a hideously tricky piece of recursive code that has been a source of trouble for days." It seems likely that reuse would reduce the joy described and thus developers for whom challenge seeking is a major motivation should reuse less existing code.

Related to the above effect of challenge seeking, reuse should also be of lower importance to developers who work on their project for the pleasure they experience when writing code *(H4b)*. Code reuse would reduce their need to write own code and thus reduce the pleasure derived from doing so. Hars and Ou (2002, p. 28) provide a nice illustration for this argumentation when they quote an OSS developer explaining his motivation to work on his project with his "innate desire to code, and code, and code until the day I die." It seems more than plausible that a developer feeling this way about coding would, ceteris paribus, reuse less. As for challenge seeking, one might argue that developers who code for fun might reuse more in order to focus on the most enjoyable tasks. However, again, this is statistically controlled for by including developers' perception of task selection benefits through reuse (see *H1d* above).

The goal to improve one's software development skills could affect reuse intensity in two directions. One could conjecture that developers who want to hone

their skills purposefully reinvent the wheel in order to learn how it is done. Yet, we argue that countervailing effects dominate, such that developers for whom skill improvement is more important also reuse more existing code *(H4c)*. Our rationale is based on DiBona's (2005) finding that OSS developers leverage existing code as a starting point for their learning and study and modify it to improve their own skills. We also found confirmation for this stance in our interviews[5] in which developers for example told us that they have "used code reuse as a way of learning" or pointed out that "reusing code snippets can really help to learn a new programming language." Also supportive to our argumentation is the finding from our survey[6] that about 50% of the developers modify the components they reuse and thus do not practice black-box reuse in which they do not get in touch with the source code of the components.

Regarding community commitment as motivation we argue that developers who feel strongly committed to the OSS community and want it to be successful will reuse more code *(H4d)*. Code reuse helps these developers to write better software faster, and allows them to make the community stronger by contributing this software.

As the last two motivations conjectured to influence developers' reuse behavior we turn to reputation building, first within the OSS community and second for the purpose of signaling skills to potential commercial partners such as employers. Regarding developers' reputation within the OSS community we argue that developers seeking to improve their reputation will reuse more code *(H4e)*. Code reuse should make a project better and thus create more attention for the project within the OSS community and also for the developers associated with the project. This argumentation receives support from Sen et al. (2008) who find that developers for whom OSS reputation building is important prefer to be part of a successful project with many other developers over being one of only a few developers of a less successful project. One could object that an OSS developer's reputation is grounded in her technical skills which she best proves with her unique—that that is, not reuse-based—contributions to the OSS community. Yet, this argumentation is refuted by von Krogh et al.'s (2003) finding that developers

[5] See the next section for an overview of our interviews.
[6] The survey is introduced in detail in the next section.

who need to prove their worthiness to join a project by making their initial contributions often include reused code in these. Furthermore, Raymond's (2001, p. 24) famous saying that "good programmers know what to write. Great ones know what to rewrite (and reuse)" also leans toward our hypothesis that developers for whom reputation building in the OSS community is important will reuse more existing code. Finally, and basically following the some argumentation as above, we posit that developers who want to signal their software development skills to potential employers or business partners will reuse more code because parties outside of the OSS community are more likely to become aware of successful OSS projects and their developers *(H4f)*. Summarizing, we posit the following hypotheses addressing the compatibility between developers' motivations to work on their project and code reuse:

Developers reuse more existing code…

H4a: …the less important challenge seeking…

H4b: …the less important coding fun and enjoyment…

H4c: …the more important skill improvement…

H4d: …the more important community commitment…

H4e: …the more important OSS reputation building…

H4f: …the more important commercial signaling…

…is for them as a motivation to work on their project.

Finally, multiple additional control variables are included in our model to account for further contextual differences in code reuse behavior. These control variables encompass four groups. First, we account for the project characteristics project size (number of project team members), technical complexity of the project, the project's position in software stack, and whether the project aims at creating a standalone executable application or a reusable component. In addition to that we further control for the level of professionalism and seriousness with which developers work on their current main project by including the number of years they

have already been involved in OSS,[7] the average weekly hours they invest into their current main project, the share of functionality that was developed by them in their current main project as compared to their project team members, and whether they have worked or work as professional software developers. Moreover, we account for developers' education and training on reuse, which has been shown to be a determinant of reuse behavior in software development firms in previous research (e.g. Frakes and Fox, 1995). Finally, we accommodate developers' geographic residence on a continent level. Subramanyam and Xia (2008) have shown that developers from different geographies prefer, for example, different levels of modularity in their OSS projects. Following this line of thought, geographic origin might also be an antecedent for reuse behavior.

4. Research Design, Data and Measures

We collected data for our study using a web-based survey that was developed based on 12 interviews with OSS developers[8] and on the existing literature. Moreover, all questionnaire items and questions were assessed for clarity by fellow researchers and OSS developers in a qualitative pretest. In the survey, we asked developers about their experiences with code reuse in the context of their current main OSS project. In order to capture the high heterogeneity of OSS projects and their developers, we chose the largest OSS project repository, SourceForge.net, as a platform to selected survey participants. In April 2009, two rounds of quantitative pretests, to which in total 2,000 developers had been invited, were conducted to assess the quality of our questionnaire in terms of content, scope, and language. Following minor refinements based on an analysis of the pretest and feedback from the respondents, the main survey took place in July 2009. An email was sent to 7,500 developers from SourceForge.net inviting them to participate in our survey. The developers were selected at random from all SourceForge.net developers who had been active on the platform in the first half of

[7] The number of years a developer has been active in OSS is treated as a control variable and not included in the local search hypotheses because it is not the intensity of experience (as e.g. measured by the number of years), but rather the breadth of experience (as e.g. measured by the number of projects involved) which is conjectured to facilitate better access to local search and consequently more code reuse.

[8] Ten of these interviews had been conducted via phone or Internet-based voice communication, the two others were conducted via email exchange. Nine of the voice-based interviews were taped and transcribed and had an average length of 49 minutes.

2009. We received a total of 686 responses, equaling a response rate of 9.6 percent (338 invitations could not be delivered). This rate is similar to those obtained by other recent surveys among SourceForge.net developers (e.g. Wu et al., 2007; Sen et al., 2008). Eleven responses had to be eliminated due to inconsistent or corrupt entries, leaving us with 675 completed surveys.

The demographic profile of the developers participating in our study (see Table 1) is largely consistent with that reported by other studies among OSS developers (e.g. Lakhani and Wolf, 2005; Sen et al., 2008). In particular, we find no indication that nonresponse has biased our sample to overrepresent less serious OSS developers.[9] Of special relevance to our endeavor is the fact that only 92 percent (or 624) of the developers we surveyed actually write code for their OSS projects. As only developers writing code can practice code reuse, our further analyses will focus on these 624 developers.

Before starting the analysis of our data, we briefly assess the multi-item constructs we have employed to measure developers' motivation to work on their main project. The items for these constructs were adopted from prior research both in the OSS domain (Hars and Ou, 2002; Lakhani and von Hippel, 2003; Roberts et al., 2006) and in psychological motivation research (Amabile et al., 1994; Clary et al., 1998), and were measured on seven-point Likert scales ("strongly disagree" to "strongly agree"). We took several steps to ensure validity and reliability of these measures. Content validity was qualitatively assessed through building on existing OSS literature whenever possible, discussions with fellow OSS researchers, and two rounds of pretests. Reliability was assessed via Cronbach's α for each multi-item variable. Not all Cronbach's α values exceed Straub's (1989) rule of thumb of 0.8, but they all exceed Nunnally's (1978) threshold of 0.6 (see Table A1 in the Appendix). Convergent validity was assessed through factor analysis, which confirms that all items have their highest loading with their respective intended

[9] Given the large number of surveys among SourceForge.net developers, one might suspect that especially the more active developers on this platform would show signs of "survey fatigue." However, comparing the self-reported weekly hours developers spend working on their main project between our survey (mean: 8.8) and the first SourceForge.net survey ever taken by Lakhani and Wolf (2005) (mean: 7.5), mitigates these concerns. The additional finding, that 69 percent of the developers in our survey have worked as professional software developers or are still working as professional software developers with an average tenure of 7.9 years rules out the further concern that only less skilled programmers took part in our survey.

construct and all loadings are higher than 0.5 (Hair et al., 2006) (see Table A1 in the Appendix). Discriminant validity is demonstrated by showing that the square root of the average variance extracted of each construct is greater than its correlations with other constructs (see Table A2 in the Appendix), thus satisfying the Fornell-Larcker criterion (Fornell and Larcker, 1981).

Table 1: Demographics of Survey Participants	
	Percentage
Age (mean: 31.8, median: 30)	
1-19	5%
20-29	42%
30-39	35%
40-49	13%
50+	5%
Residence	
North America	26%
South America	5%
Europe	54%
Asia and rest of world (RoW)	15%
Highest education level	
Non-university education	15%
Undergraduate or equivalent	35%
Graduate or equivalent	30%
Ph.D. and higher	20%
Task profile in open source projects	
Includes writing code	93%
Does not include writing code	7%
Hours spent working on main OSS project per week (mean: 8.8, median: 5)	
1-4	48%
5-9	19%
10-19	21%
20+	12%
Size of personal OSS network (mean: 29.9, median: 8)	
1-9	70%
10-19	18%
20+	12%
Number of OSS projects ever involved in (mean: 3.7, median: 2)	
1-4	65%
5-9	26%
10-14	6%
15+	3%

In order to reduce common method bias, we employed several measures during data collection as suggested by Podsakoff et al. (2003). We have taken care to formulate simple and unambiguous questions for our survey by discussing our questionnaire items with our interview partners and conducting multiple rounds of pretests. Further, survey respondents were assured when the survey was

introduced to them that their responses would be treated strictly confidentially. Moreover, much of the survey items address motivations, attitudes, and beliefs for which by nature there are no right or wrong answers.

To estimate the presence of common method bias in our data after survey completion, we employed Harman's test in which all variables of a model are loaded onto a single factor in a principal component factor analysis. A significant amount of common method bias is assumed to exist if this one factor explains a large portion of all the variance in the data (Podsakoff et al., 2003). In our data we find the maximum variance explained by one factor being 9.3 percent, which does not hint toward strong common method bias.

5. Results and Discussion

Following the research questions presented above, this section consists of four parts. In the first, we establish the importance of code reuse in OSS development. Next, we present perceived benefits and issues of reuse as well as impediments to it, and address the question of why OSS developers do or do not reuse code. The third part presents the core of this study in the form of a multivariate analysis of code reuse behavior used to test our research model. In the final, fourth part we discuss potential threats to validity and limitations of our study.

5.1. Importance of Code Reuse

When measuring code reuse we focused on component and snippet reuse. In our survey, component reuse was defined as "*reusing of functionality from external components in the form of libraries or included files. E.g., implementing cryptographic functionality from OpenSSL or functionality to parse INI files from an external class you have included. Please do not count functionalities from libraries that are part of your development language, such as the C libraries.*" In a similar fashion, snippet reuse was defined as "*reusing of snippets (several existing lines of code) copied and pasted from external sources. If you have modified the code after copying and pasting it by, e.g., renaming variables or adjusting it to a specific library you use, this would still be considered as […] reuse […].*"

Three different measures (depicted in Table 2) were employed to investigate the importance of code reuse. First, related to, for example, Cusumano

and Kemerer (1990) or Frakes and Fox (1995), we asked developers to indicate the share of functionality based on reused code that was added by them to their current main project. We found that, on average, nearly one third (mean=30%, median=20%) of the functionality OSS developers have added to their project was based on reused code, which points out that code reuse is indeed an important element of OSS development. This interpretation is further supported by the fact that only six percent of the developers surveyed report that they have not reused any code at all. Furthermore, the maximum share of reused functionality of 99 percent shows that some developers rely very heavily on code reuse and see their role mainly in writing "glue-code" to integrate the various pieces of reused code. As a second measure, we employed a self-developed four-item scale to directly measure the perceived importance of reuse for the individual developers' work on their main project.[10] On seven-point Likert scales, developers indicated their agreement to four statements that described, in various ways, reuse as "very important." With a mean of 4.74 (median=5.25) and 58 percent of all developers at least "somewhat agreeing" to the statements, the important role of code reuse in OSS development is again confirmed.

Finally, as the third approach, using a further self-developed four-item scale,[11] we asked developers to indicate their intent to reuse existing code in the future development of their current main project. The results are largely similar to those obtained by using the second measure (perceived importance of reuse in past work), once more indicating that code reuse is very important. However, both mean and median are significantly lower (mean=4.57, median=4.75) than in the previous

[10] The scale was developed based on our interviews with developers and on research on general knowledge reuse (Watson and Hewett, 2006). It also draws on the intention and behavior scales commonly employed in TAM or TPB research in the IS domain, for example, by Riemenschneider et al. (2002) or by Mellarkod et al. (2007). The statements of the scale are: "Reusing has been extremely important for my past work on my current main project," "Without reusing, my current main project would not be what it is today," "I did reuse very much during my past work on my current main project," and "My past work on my current main project would not have been possible without reusing." The scale explains 83.4 percent of the total variance and Cronbach's α is 0.93.

[11] The statements of the scale are: "Reusing will be extremely important in my future work on my current main project," "Realizing my future tasks and goals for my current main project will not be possible without reusing," "I will reuse very much when developing my current main project in the future," and "Realizing my future tasks and goals for my current main project will be very difficult without reusing." The scale explains 83.8 percent of the total variance and Cronbach's α is 0.94.

measure. This finding might be a first indication supporting hypothesis *H3*, which states that code reuse is more important in earlier phases of an OSS project.

Table 2: Average Share of Functionality Reused by Developer (in %)					
Measure	Mean	Median	S.D.	Min.	Max.
Share of implemented functionality based on reused code (in %)	30.0%	20.0%	26.5%	0.0%	99.0%
Importance of reuse for past work on project (seven-point Likert scale)*	4.74	5.25	1.86	1.00	7.00
Importance of reuse for future work on project (seven-point Likert scale)*	4.57	4.75	1.69	1.00	7.00
*Measure is based on four single items. N=624.					

Despite the prominent role of code reuse as consistently indicated by all three measures, the high standard deviations also reveal large heterogeneity in developers' code reuse behavior. Developers' individual reasons for and against code reuse in their development are suspected to largely drive this heterogeneity and will be explored in the following section.

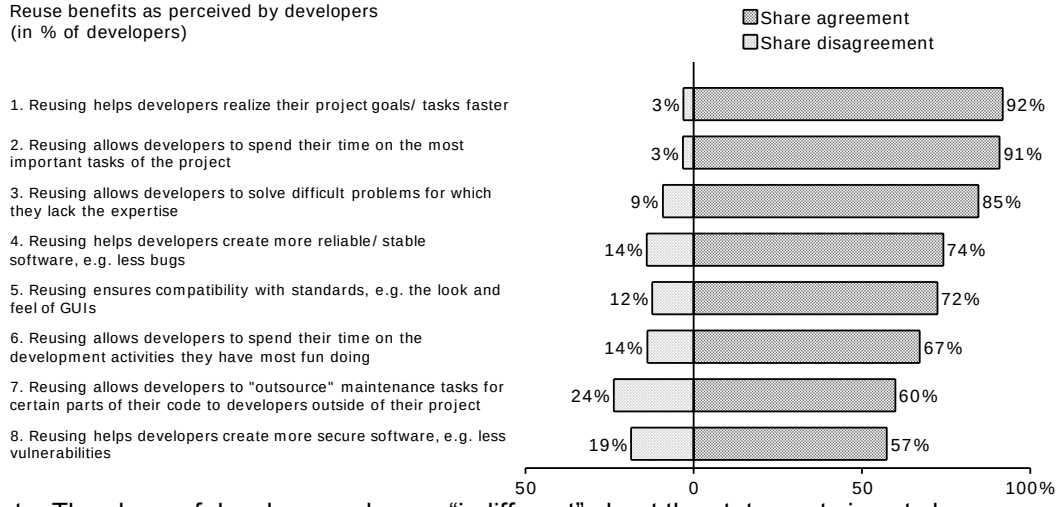
5.2. Developers' Reasons For and Against Code Reuse

In our analysis of developer's reasons for and against code reuse, we differentiate between three sets of factors. First, we analyze the benefits of code reuse as perceived by OSS developers. Second, we investigate the drawbacks and issues that developers see in code reuse, and, finally, we address the importance of general impediments[12] to code reuse.

Based on our interviews, as well as the existing literature, we have identified eight distinct benefits of code reuse. Survey participants were asked to indicate their agreement on a seven-point Likert scale to statements regarding these benefits. Results are displayed in Figure 2 and show that all of the statements received rather high shares of agreement. The two statements with the highest level of agreement both point to efficiency effects of reuse, followed by a statement pertaining to its effectiveness effects. For the benefits on ranks four and higher, agreement drops significantly compared to rank three, yet is still quite high. Ranked fourth and fifth are statements addressing effects of reuse on the quality of the

[12] While these "general impediments" are rather objective compared to developers' beliefs about benefits and issues, they may still reflect individual developer's opinions, having been measured by asking the developers.

software being developed by making it more stable and more compatible with standards. The statement ranked eighth, about the effects of code reuse on software security also pertains to this group, however, it receives considerably less agreement. This could be explained by the fact that many OSS projects develop types of software for which security is not a major concern, for example, games. Ranked sixth and seventh are statements that position reuse as a means for developers to select their project tasks by preference and avoid mundane jobs. An example for this is "outsourcing" maintenance work to the original developers of the reused code who fix bugs or implement new functionality in the code of which the reusing developer benefits without having to do this work by herself.



Note: The share of developers who are "indifferent" about the statements is not shown. N=624.

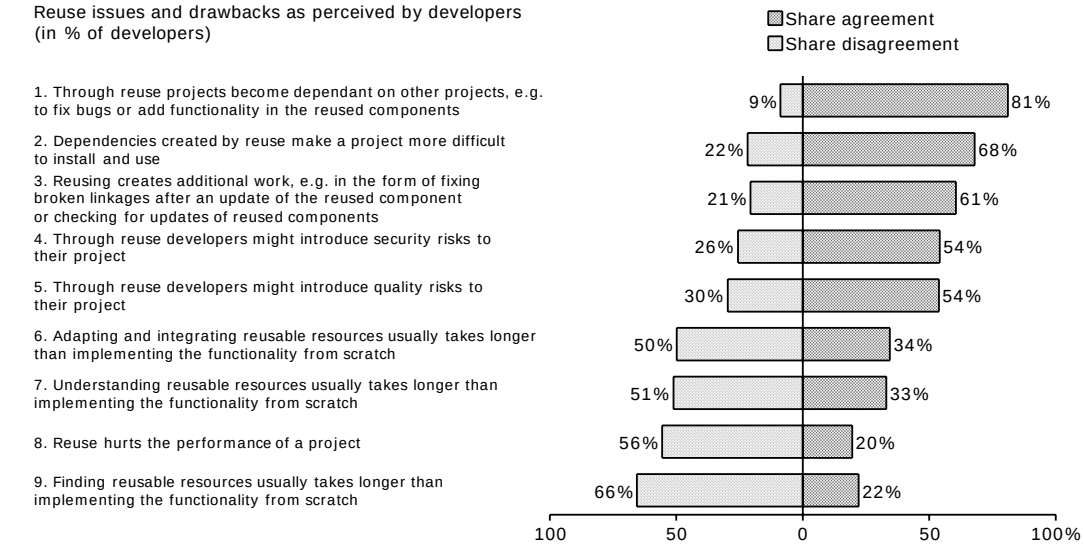
Figure 2: Share of Developers that Disagree/Agree to Reuse Benefits

In order to check consistency of responses and to construct factor scores to be used in the multivariate analyses later, an exploratory factor analysis is carried out. With four components, it explains 77.2 percent of total variance and yields good quality measures (KMO: 0.76, $p<0.0001$).[13] The resulting components can be interpreted as development efficiency (ranks 1, 2), software quality (ranks 4, 5, 8), task selection (ranks 6, 7), and development effectiveness (rank 3).[14]

[13] For better interpretability of the resulting components, components with an Eigenvalue of less than 1 were also extracted. The fourth component had an Eigenvalue of 0.79.

[14] The factor analysis uses principal component analysis and Varimax rotation. Cronbach's α for the components software quality, development efficiency, and task selection is 0.80, 0.72 and 0.47, respectively. See Table A3 in the Appendix for detailed factor loadings.

Following the benefits of code reuse, nine issues and drawbacks identified in our interviews and existing literature (shown in Figure 3) were presented to participants who were again asked to indicate their agreement to the respective statements. The highest share of agreement was received by a statement pointing to the loss of control that a developer may have to accept when reusing code. Statements ranked second and third also relate to losing control, however, with significantly lower levels of agreement. The statement ranked second points to software being more difficult to install (build) and use by end-users due to technical dependencies, while the statement ranked third reflects the developer's obligation to check and integrate updates of reused code.[15] Ranked fourth, fifth, and eighth—and again with significantly lower levels of agreement than the previous statements—are two potential issues of code reuse that point to quality and security risks. The statements ranked sixth, seventh, and ninth all describe situations where development from scratch is more efficient than code reuse. They do, however, receive at least 50 percent disagreement, which emphasizes that most developers do not deem searching, understanding, and adapting reusable code as inefficient.



Note: The share of developers who are "indifferent" about the statements is not shown. N=624.

Figure 3: Share of Developers that Disagree/Agree to Reuse Issues and Drawbacks

[15] Both statements mainly refer to component reuse and are only partially applicable to snippet reuse.

An exploratory factor analysis of these issues and drawbacks explains 69.0 percent of total variance with three components, and yields good quality measures KMO: 0.72, p<0.0001). The resulting components can be interpreted as control loss (ranks 1, 2, 3), quality risks (ranks 4, 5, 8), and inefficiency of reuse (ranks 6, 7, 9).[16]

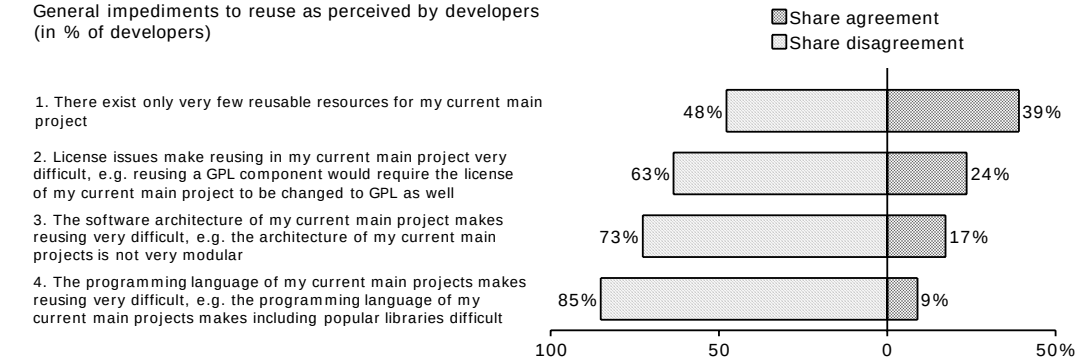
To consolidate the number of variables in the multivariate model employed later, a further factor analysis merged the software quality benefits and the quality risks into one component. Further, the development efficiency benefits were merged with the inefficiency of reuse. The five final components used in the multivariate model are: effectiveness benefits, efficiency benefits, quality benefits, task selection benefits, and loss of control risks.

While the benefits and issues/drawbacks of code reuse were subjective and perceived by the individual developer, there also exist general impediments to reuse. These general impediments which resulted from our interviews and existing literature make code reuse difficult or impossible even if the individual developer wanted to rely on existing code (see Figure 4). Interestingly, however, all four statements offered to the surveyed developers received more disagreement than agreement. The statement "*there exist only very few reusable resources for my current main project"* ranked first, with 39 percent of the developers agreeing. Oneway ANOVA analysis used to identify for which projects there exist least reusable resources found only the target operating system of a project having a significant influence on the availability of reusable code (p=0.0497). Projects that are not developed for POSIX operating system systems (e.g., Linux) or Windows have less reusable code at their disposal. Neither the type of the project (e.g., "Software Development," "Scientific and Engineering," or "Games and Entertainment") had any significant influence (p=0.2440), nor did the graphical user interface employed by the project (0.1171).

Ranked as the second general impediment to code reuse with 24 percent agreement are license incompatibilities. Such a situation would occur, for example, if a programmer wanted to reuse code snippets licensed under the GPL in a project licensed under the BSD license. As expected, the license of the developer's main project significantly influences this general impediment (Oneway ANOVA,

[16] The factor analysis uses principal component analysis and Varimax rotation. Cronbach's α for the components control loss, quality risks, and inefficiency of reuse is 0.66, 0.76 and 0.85, respectively. See Table A4 in the Appendix for detailed factor loadings.

p<0.0001), with developers working on GPL licensed projects least likely to perceive this as an issue. However, the low share of agreement is surprising. Three possible explanations for this finding seem plausible: First, there might exist enough reusable code in each license category. Second, developers might able to mitigate the license incompatibilities through modular project architectures that clearly separate modules under different licenses and thus avoid contamination issues (Henkel and Baldwin, 2009). Third, developers are not knowledgeable about license incompatibilities and ignore the potential issues. Ranked third and fourth with 17 percent and nine percent agreement, respectively, are the architecture of the developer's current main project being not modular enough to allow for easy integration of reusable code (rank 3) and incompatibilities between the project's main programming language and the programming language of the code the developer wants to reuse (rank 4). Both are significantly dependent on the programming language of the developer's project (Oneway ANOVA, p=0.0036 and p<0.0001 for rank 3 and rank 4, respectively), with C++ and Java as object-oriented languages posing the least issues.



Note: The share of developers who are "indifferent" about the statements is not shown. N=624.

Figure 4: Share of Developers that Disagree/Agree to General Reuse Impediments

5.3. Multivariate Analysis of Reuse Behavior

Following the descriptive analysis, the objective of our research model is to explain the observed heterogeneity in developers' reuse behavior found earlier with both developer and project characteristics. We test the research model with our three different measures of reuse behavior as dependent variables in three different

regression models in order to ensure robustness of results.[17] All three models will be tested using Tobit regressions as their dependent variables are restricted to either [0-100%] or [1-7].[18] A summary of the research model hypotheses and the support they received in the multivariate analyses is presented in Table 3 while the detailed regression tables containing the Tobit models are depicted in Table 4. As a further robustness check, we ran specifications of the three models with successive elimination of insignificant variables. The results of this robustness check which are largely consistent with the results of the main models are shown in Table A7 in the Appendix. The results of the multivariate analyses are presented and discussed in the following.

Table 3: Summary of hypotheses testing

Hypotheses	Confirmed?
Attitude toward reuse: Developers reuse more on existing code…	
H1a: …the more strongly they perceive the effectiveness benefits of reuse.	✓
H1b: …the more strongly they perceive the efficiency benefits of reuse.	✓
H1c: …the more strongly they perceive the quality benefits of reuse.	✓
H1d: …the more strongly they perceive the task selection benefits of reuse.	✓
H1e: …the less strongly they perceive the loss of control risks of code reuse.	✗
Access to local search: Developers reuse more existing code…	
H2a: …the larger their personal OSS network.	✓
H2b: …the greater the number of OSS projects they have been involved in.	✓
Project maturity:	
H3: Developers reuse more existing code the less mature their project.	✓
Compatibility with project goals: Developers reuse more existing code…	
H4a: …the less important challenge seeking is for them as a motivation to work on their project.	✓
H4b: …the less important coding fun and enjoyment is for them as a motivation to work on their project.	✗
H4c: …the more important skill improvement is for them as a motivation to work on their project.	✗
H4d: …the more important community commitment is for them as a motivation to work on their project.	✓
H4e: …the more important OSS reputation building is for them as a motivation to work on their project.	✗
H4f: …the more important commercial signaling is for them as a motivation to work on their project.	✗

Legend: ✓: fully confirmed; ✓: partially confirmed; ✗: not supported

[17] Descriptive statistics of all explanatory variables are depicted in Table A5 in the Appendix. The correlation matrix is shown in Table A6 in the Appendix.

[18] In contrast to an OLS regression, a Tobit model accounts for the censoring of the dependent variable. In the present case this means, for example, that the share of functionality from reused resources cannot be less than zero percent, or larger than 100 percent.

Table 4: Multivariate Analysis of Developers' Reuse Behavior			
	Past importance of reuse		(3) Future importance of reuse (Likert scale)
	(1) Likert scale	(2) Percentage scale	
Attitude toward reuse			
BenefitEffectiveness *(H1a)*	0.222*** (0.076)	2.701*** (1.021)	0.168*** (0.063)
BenefitEfficiency *(H1b)*	0.653*** (0.084)	5.959*** (1.114)	0.517*** (0.069)
BenefitQuality *(H1c)*	0.303*** (0.081)	1.800* (1.073)	0.250*** (0.067)
BenefitTaskSelection *(H1d)*	0.155** (0.078)	3.528*** (1.041)	0.132** (0.064)
IssueControlLoss *(H1e)*	-0.030 (0.077)	-0.506 (1.036)	-0.004 (0.064)
Access to local search			
DevOSSNetsize (log) *(H2a)*	0.165** (0.083)	2.098* (1.102)	0.230*** (0.069)
DevOtherProjects *(H2b)*	0.022 (0.016)	0.398* (0.208)	0.032** (0.013)
Project maturity			
ProjPhase *(H3)*	-0.149** (0.070)	-3.227*** (0.928)	-0.219*** (0.057)
Compatibility with project goals			
MotChallenge *(H4a)*	-0.148* (0.083)	-2.559** (1.103)	-0.067 (0.068)
MotFun *(H4b)*	0.098 (0.080)	0.575 (1.072)	0.055 (0.066)
MotLearning *(H4c)*	0.003 (0.080)	-1.438 (1.053)	-0.015 (0.066)
MotCommunity *(H4d)*	0.177** (0.086)	1.964* (1.150)	0.148** (0.071)
MotOSSReputation *(H4e)*	0.005 (0.057)	0.128 (0.758)	0.065 (0.047)
MotSignaling *(H4f)*	-0.054 (0.061)	0.336 (0.817)	0.013 (0.051)
Subjective norms			
DevNorm	0.140** (0.066)	2.372*** (0.887)	0.197*** (0.055)
Perceived behavioral control			
ProjPolSupport	0.440** (0.200)	0.946 (2.670)	0.297* (0.165)
ProjPolDiscourage	-1.087** (0.457)	-4.977 (6.161)	-1.279*** (0.383)
ConditionLack	-0.250*** (0.044)	-2.317*** (0.589)	-0.168*** (0.036)
ConditionLicense	0.065 (0.045)	0.309 (0.599)	0.018 (0.037)
ConditionLanguage	0.030 (0.060)	-0.071 (0.802)	0.060 (0.049)
ConditionArchitecture	0.017 (0.052)	0.481 (0.698)	0.017 (0.043)
DevSkill	-0.075 (0.095)	-0.123 (1.270)	-0.018 (0.078)
Further control variables			
ProjSize	0.000 (0.002)	-0.021 (0.024)	-0.002 (0.001)
ProjComplexity	0.131 (0.092)	2.194* (1.236)	0.0190 (0.076)
ProjStack	0.210** (0.091)	1.499 (1.209)	0.135* (0.074)
ProjStandalone	0.118 (0.197)	0.233 (2.633)	0.203 (0.163)
DevOSSExperience	0.010 (0.018)	0.076 (0.249)	0.000 (0.015)
DevProjTime	0.014* (0.008)	-0.039 (0.107)	0.008 (0.007)
DevProjShare	0.003 (0.002)	0.031 (0.033)	0.001 (0.002)
DevProf	0.056 (0.186)	0.214 (2.492)	0.184 (0.154)
DevEduReuse	-0.127 (0.165)	-1.177 (2.201)	-0.266* (0.136)
DevProfEduReuse	0.603** (0.237)	5.883* (3.094)	0.378* (0.193)
Residence-N. America	-0.159 (0.181)	-3.310 (2.408)	0.120 (0.149)
Residence-S. America	0.236 (0.359)	-3.424 (4.743)	-0.013 (0.294)
Residence-Asia & RoW	-0.102 (0.226)	0.764 (3.031)	-0.109 (0.187)
Constant	3.026*** (0.888)	23.275* (11.87)	2.545*** (0.731)
Observations	624	624	624
Pseudo R^2	0.107	0.029	0.119
Likelihood ratio	$X^2(35)=267.42$, p<0.0001	$X^2(35)=162.74$, p<0.0001	$X^2(35)=289.55$, p<0.0001
σ	1.790	24.337	1.493

Notes: All models are Tobit models; standard errors in parentheses; * significant at 10%; ** significant at 5%; *** significant at 1%.

5.3.1. Attitude Toward Reuse

The regression results confirm hypotheses *H1a* to *H1d*. Developers who perceive higher effectiveness, efficiency, quality, or task selection benefits from code reuse attribute a higher importance to it and practice it more. The coefficients for all four hypotheses are positive and significant for all dependent variables and all specifications. In contrast, hypothesis *H1e* is not confirmed. The data does not show that developers who fear to lose control over their project reuse less code. This is surprising as, in our descriptive analysis, loss of control was ranked as the main issue developers have with code reuse. A plausible interpretation is that developers' concerns about losing control over their project affect their decision as to which code to reuse, but do not affect the total amount of code they reuse. For example, developers concerned about losing control might choose to reuse only components developed by other projects that have a proven track record of fixing bugs quickly and keeping the structure of their code stable (Haefliger et al., 2008).

5.3.2. Access to Local Search

The effect of developers' access to local search on their reuse behavior was captured by the logarithm of the size of their OSS network *(H2a)* and the number of other OSS projects they have been involved in *(H2b)*. Hypothesis *H2a* is confirmed in all models while *H2b* is confirmed only partially, its coefficient not being significant in model 1. Nonetheless, all coefficients are positive in all models, supporting our assumption that developers that can access, evaluate, understand, and integrate reusable code more easily due to local search practice more code reuse.

The finding that the number of years a developer has been involved in OSS does not exhibit a significant effect on her reuse behavior (see control variable DevOSSExperience) is consistent with our argumentation regarding local search. We had claimed that developers who can turn to their personal OSS network or their experience in other OSS projects reuse more because of their better access to local search. A greater number of years involved in OSS alone does not yet facilitate such better access because for example a developer with ten years of OSS work spent in only one project does not have access to local search regarding which code other projects use to solve a particular problem.

5.3.3. Project Maturity

Our hypothesis that developers reuse less code once their project has matured *(H3)* is confirmed across all dependent variables and specifications.[19] Developers do indeed seem to leverage reuse as a tool to deliver a "credible promise" early on and overcome liabilities of newness to get on a par with competing existing projects, while later project phases call for specific refinements of their projects where there is less available code to reuse.

5.3.4. Compatibility with Project Goals

Regarding the compatibility of code reuse with a developer's individual project goals, hypothesis *H4d* (community commitment) is confirmed in all models except model 2; *H4a* (challenge seeking) is confirmed only in models with past reuse as the dependent variable (models 1, 2 and 5). For all other hypotheses (coding fun and enjoyment *(H4b)*, skill improvement *(H4c)*, OSS reputation building *(H4e)*, and commercial signaling *(H4f)*) the null hypothesis cannot be rejected.

The support for hypothesis *H4d* highlights that developers who feel they are part of the OSS community and want it to grow and be successful rely more on code reuse than other developers. Code reuse is compatible with their goal of contributing to the OSS community because by leveraging code reuse they can contribute more and in higher quality.[20] The partial confirmation of hypothesis *H4a* supports our assumption that the developers' goal to seek and tackle technical challenges impedes code reuse. By reusing existing code, developers would not be denied the pleasure of solving a problem by themselves. Thus, they would rather refrain from code reuse if challenge seeking is of major importance to them in their OSS work. The finding that the respective coefficient is not significant when the dependent variable is the developers' future intent to reuse may be due to the desire

[19] Note that in models 1, 2, 4 and 5 where past reuse behavior is the dependent variable, the amount of reused code reported by developers with projects in later development phases is their average reuse level including the assumed high levels of code reuse of early phases and the proposed lower levels of later phases. However, if reuse goes down with maturity as proposed then also average reuse decreases over the lifetime of a project.

[20] Moreover, developers who are more sympathetic toward the OSS community might also be affected by the general positive attitude toward reuse of this community (e.g. Raymond, 2001). This effect is, however, captured via subjective norms as control variable.

developers may have to solve a problem by themselves, without external help, is something that can occur spontaneously and is thus difficult to predict.

We now turn to those hypotheses that are not supported. We had argued that similarly to challenge seeking, the fun and enjoyment developers experience when writing code leads them to reuse less code *(H4b)*, but we cannot confirm this hypothesis. In fact, the respective coefficients are not negative as expected, but positive, though insignificant. The remaining unconfirmed hypotheses, skill improvement *(H4c)*, OSS reputation building *(H4e)* and commercial signaling *(H4f)* partially show varying coefficient signs. This could be because, contrary to our assumptions, code reuse could be both supportive as well as detrimental to these goals. While reused code could be used as an example to improve programming skills, it could also hamper learning if developers treat the reused code as a black box. Regarding reputation building and commercial signaling, we had expected that developers who make their projects more successful with the help of code reuse are regarded more highly in the OSS community and can present themselves as better developers to potential employers or business partners. However, it is also possible that in certain situations the code created by developers themselves without the help of code reuse is important to build their OSS reputation or signal skills to potential employers and partners. In these situations, developers would refrain from code reuse if reputation building or signaling is a main motivation for their OSS work.

5.3.5. Control Variables

Due to the large number of control variables included in our model, we only point out a few main results. The social norms as perceived by developers show a consistently significant and positive influence as predicted by TPB. Consequently, OSS developers who feel that their peers appreciate them reusing existing code will reuse more. Of the variables describing developers' perceived behavioral control, the lack of reusable code has a consistently negative and significant influence on reuse behavior. With the exception of one dependent variable, project policies discouraging reuse lead to reduced code reuse, while policies promoting reuse are found to significantly increase reuse behavior in three models (1, 4, 6). Lastly, developers who had received training on reuse in companies, practice significantly more code reuse, while developers who had only learned about reuse during their

academic education do not differ in their code reuse behavior from developers who had not had reuse in their curriculum.

To summarize, the regression analyses shed light on developers' code reuse behavior. In particular, the (partially) confirmed hypotheses *H2* (access to local search), *H3* (project maturity), and *H4a* (challenge seeking) provide interesting findings that are also relevant beyond the scope of OSS.

5.4. Possible threats to validity and limitations of the study

In the following we employ the four generally accepted criteria of validity (Cook and Campbell, 1979) as our structure: Construct validity, internal validity, statistical conclusion validity and external validity.

Construct validity threats concern the ability to measure what we are interested in measuring. As pointed out in sections 4 and 5, the measures employed in this study are based on existing measures from other studies and our interviews. All measures were assessed for clarity by other researchers and OSS developers during pretests as described above. Furthermore, all multi-item constructs were quantitatively gauged with regards to reliability, convergent validity, and discriminant validity. We thus consider our study to possess sufficient construct validity. Nonetheless, a potential issue is whether developers are able to accurately estimate their level of code reuse in a questionnaire. However, while an additional verification of our results using an objective measure of code reuse is certainly worthwhile, developers in our pretests convinced us that they can, with considerable precision, estimate their degree of code reuse. Furthermore, to ensure robustness of our findings we have employed three different measures of code reuse in the survey. Finally, also many other reuse studies rely on reported reuse levels (e.g. Frakes and Fox, 1995; Lee and Litecky, 1997).

Internal validity, maintaining that there should not exist alternative explanations for the relationships identified between our research model constructs, should also be given since our research model relies on the well established TPB and because we have included multiple further control variables derived from our interviews and OSS or reuse literature. A potential issue is our approach to deal with component and snippet reuse simultaneously. If component reuse in OSS development equaled black-box reuse there might exist different drivers for it than

for snippet reuse. However, because we find that about 50% of the surveyed developers modify the components they reuse we argue that at least in the OSS context component reuse does not constitute typical black-box reuse. Consequently, we expect both component and snippet reuse to be influenced by largely the same drivers.

In addition to that, we also consider our results to be *valid with regard to our statistical conclusions* since they are based on a sample of considerable size and backed by the significance levels of our hypotheses as well as the largely consistent results in various model specifications and with various dependent variables.

Finally, *external validity* threats concern the generalization of our findings. In line with the other main studies of individual OSS developers we drew our sample from SourceForge.net developers. As pointed out in chapter 4, we have no reason to believe that our sample is not representative of SourceForge.net developers. Thus, generalization for this most frequently researched group of OSS developers should be feasible. To ensure external validity when generalizing to OSS developers registered on other platforms (where e.g. projects are larger) or to traditional software developers working on proprietary software in commercial firms it would be necessary to replicate our study in these settings. However, both our data as well as our research model suggest that generalization to other contexts should yield similar results. For example on the data side we do not find significant differences between the reuse behavior of paid and hobbyist OSS developers. Regarding the research model, it would be surprising to find that rather general hypotheses such as the effect of network size or challenge seeking work differently in the context of proprietary software development.

6. Conclusion

In this paper, we set out to use quantitative data obtained through a survey to explain and understand code reuse in OSS projects. Contributing to the emerging stream of scholarly work on code reuse in OSS, we present strong evidence that code reuse is of major importance in OSS development and has contributed to its success. We further show that OSS developers perceive efficiency and effectiveness as the main benefits of code reuse. Of relevance not only to OSS research but also to the domains of software engineering and the receiving side of

open innovation processes in general, our investigation of drivers of code reuse finds that developers with better access to local search due to a larger personal OSS network or more exposure to different OSS projects reuse more existing code, presumably because their costs of accessing this code are lower. Further, developers convinced of the benefits of code reuse (efficiency and effectiveness gains, enhanced software quality, and the chance to work on preferred tasks) practice it more, as do developers who can use code reuse to support their goal of serving the OSS community. Moreover, developers see code reuse as a means to kick-start new projects as it helps them deliver a "credible promise" and close the gap to existing and competing projects more quickly. Lastly, we find partial support for our hypothesis that those developers who desire to solve technical problems for the satisfaction of it refrain from reuse and, thus, make their projects less efficient and effective than they could be.

As academic work on code reuse in OSS has only just begun, it merits further research. While our study has addressed development with reuse, future work should investigate development for reuse, that is OSS projects which develop components primarily intended to be reused in other projects. Questions of relevance in this context are: why do developers bear the reportedly large additional costs of writing reusable code,[21] or have they have found ways to mitigate them. Additionally, as has been pointed out by Haefliger et al. (2008), the strategies that OSS developers employ to make their reusable code known and reused deserve investigation. Moreover, the limitations of our work open up several further research avenues. First, our dependent variables reflect developers' subjective perception of the importance of code reuse for their OSS work. In an alternative way, and potentially adding robustness to our findings, the importance of reuse could be captured more objectively by analyzing the code of a project. Similarly, independent variables captured from other data sources could be added to our model. For example, social network data derived from SourceForge.net (e.g. Fershtman and Gandal, 2009) could be employed to further extend and test our hypotheses on local search. Moreover, we have described code reuse in general, not differentiating between its various forms (components, snippets, algorithms). A more fine-grained analysis using these dimensions might yield further insights into the mechanics of

[21] For example Tracz (1995) estimates that writing reusable code leads to 100 percent of additional effort.

code reuse in OSS projects. Finally, while we have focused on developers and their projects as determinants of code reuse, future work could employ an even more detailed approach and analyze single reuse incidents, incorporating developers, their projects, and the artifacts they consider for reuse. Such an approach could, for instance, analyze the impact of the quality of the relationship between the "giving" and the "receiving" side of the open innovation process on code reuse.

Beyond their scholarly implications, our findings are also of relevance to managerial practice. They highlight the high level of reuse within the OSS community that should provide motivation to firms to also leverage existing OSS code in their software development, thereby partly mitigating the typically high upfront investment costs of building an internal reuse library for artifacts that are not firm-specific (Frakes and Kang, 2005).[22] Further, if they intend to pursue this avenue of reusing OSS code, commercial firms should encourage and support their employees to enhance their access to local search for OSS code by building personal OSS networks and by becoming involved in various OSS projects. Beyond reuse of OSS code, modified incentives and development processes based on our findings could support internal corporate reuse activities in software engineering and beyond. As part of such modifications, developers could be provided with the option to select tasks themselves, according to their preference, they could be compensated according to their work results delivered and not based on the time they have spent at work, and they could be required to deliver "credible promises" in new development projects (Haefliger et al., 2008). Lastly, to accommodate the desire of developers to tackle difficult technical challenges, which makes them reuse less than they could, firms could consider job enrichment (e.g. Herzberg, 1968) to integrate challenges into developers' work that are in the best interest of the firm, thereby accommodating the needs of both developer and firm.

[22] Obviously this has to be in accordance with the licenses of the OSS code. However, well-designed product architectures can mitigate many of the issues potentially arising here (Henkel and Baldwin, 2009).

7. References

Ajzen, I. (1991) "The Theory of Planned Behavior," *Organizational Behavior and Human Decision Processes* 50 (2), pp. 179-211.

Ajzen, I. (2002) "*Constructing a TpB Questionnaire: Conceptual and Methodological Considerations,"* Manuscript, University of Massachusetts, Available at URL: http://people.umass.edu/aizen/pdf/tpb.measurement.pdf.

Aldrich, H. and E. Auster (1986) "Even Dwarfs Started Small: Liabilities of Age and Size and Their Strategic Implications," in Cummings, L. and B. Staw (Eds.) *Research in Organizational Behavior*, San Francisco, CA: JAI Press, pp. 165-198.

Amabile, T.M., K.G. Hill, A. Hennessey, and E.M. Tighe (1994) "The Work Preference Inventory: Assessing Intrinsic and Extrinsic Motivational Orientations," *Journal of Personality and Social Psychology* 66 (5), pp. 950-967.

Armitage, C. and M. Conner (2001) "The Theory of Planned Behavior," *British Journal of Social Psychology* 40 (4), pp. 471-499.

Baldwin, C.Y. and K.B. Clark (2006) "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?," *Management Science* 52 (7), pp. 1116-1127.

Banker, R.D., R.J. Kauffman, and D. Zweig (1993) "Repository Evaluation on Software Reuse," *IEEE Transactions of Software Engineering* 19 (4), pp. 379-389.

Bonaccorsi, A., S. Giannangeli, and C. Rossi (2006) "Entry Strategies under Competing Standards: Hybrid Business Models in the Open Source Software Industry," *Management Science* 52 (7), pp. 1085-1098.

Chang, H.-F.A. and A. Mockus (2008) "Evaluation of Source Code Copy Detection Methods on FreeBSD," *International Working Conference on Mining Software Repositories*, Leipzig, Germany.

Chesbrough, H.W. (2003) *Open Innovation. The New Imperative for Creating and Profiting from Technology*. Boston, MA: Harvard Business School Press.

Clary, E.G., M. Snyder, R.D. Ridge, J. Copeland, A.A. Stukas, and J. Haugen (1998) "Understanding and Assessing the Motivations of Volunteers: A Functional Approach," *Journal of Personality and Social Psychology* 74 (6), pp. 1516-1530.

Cook, T.D. and D.T. Campbell (1979) *Quasi-Experimentation: Design and Analysis Issues for Field Setting*. Chicago, IL: Rand McNally.

Crowston, K. and B. Scozzi (2008) "Bug Fixing Practices within Free/Libre Open Source Software Development Teams," *Journal of Database Management* 19 (2), pp. 1-30.

Crowston, K., K. Wei, J. Howison, and A. Wiggins (2009) "*Free/Libre Open Source Software Development: What We Know and What We Do Not Know,"* (07.07.2009), Working Paper, Available at URL: http://floss.syr.edu/StudyP/Review%20Paper_070709.pdf.

Csíkszentmihályi, M. (1990) *Flow: The Psychology of Optimal Experience*. New York, NY: Harper and Row.

Cusumano, M. and C. Kemerer (1990) "A Quantitative Analysis of U.S. And Japanese Practice in Software Development," *Management Science* 36 (11), pp. 1384-1406.

Dahlander, L. (2005) "Appropriation and Appropriability in Open Source Software," *International Journal of Innovation Management* 9 (3), pp. 259-285.

Davis, F.D., R.P. Bagozzi, and R.P. Warshaw (1989) "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models," *Management Science* 35 (8), pp. 982-1002.

Desouza, K.C., Y. Awazu, and A. Tiwana (2006) "Four Dynamics for Bringing Use Back into Software Reuse," *Communications of the ACM* 49 (1), pp. 96-100.

DiBona, C. (2005) "Open Source and Proprietary Software Development," in DiBona, C., D. Cooper, and M. Stone (Eds.) *Open Source 2.0: The Continuing Evolution*, Sebastopol, CA: O'Reilly Media.

DiBona, C., J. Ockerbloom, and M. Stone (1999) "Introduction," in DiBona, C., S. Ockman, and M. Stone (Eds.) *Open Sources: Voices of the Open Source Revolution*, Sebastopol, CA: O'Reilly & Associates, pp. 1-17.

Fershtman, C. and N. Gandal (2009) "*R&D Spillovers: The 'Social Network' of Open Source,"* (16.05.2009), Working Paper, Available at URL: http://www.tau.ac.il/~gandal/OSS.pdf.

Fornell, C. and F. Larcker (1981) "Evaluating Structural Equation Models with Unobservable Variables and Measurement Error," *Journal of Marketing Research* 13 (1), pp. 39-50.

Frakes, W.B. and C.J. Fox (1995) "Sixteen Questions About Software Reuse," *Communications of the ACM* 38 (6), pp. 75-87.

Frakes, W.B. and K. Kang (2005) "Software Reuse Research: Status and Future," *IEEE Transactions of Software Engineering* 31 (7), pp. 529 - 536

German, D.M. (2007) "Using Software Distributions to Understand the Relationship among Free and Open Source Software Projects," *4th International Workshop on Mining Software Repositories*, Minneapolis, MN.

Ghosh, R.A., R. Glott, B. Krieger, and G. Robles (2002) "Free/Libre and Open Source Software: Survey and Study - Deliverable D18: Final Report - Part IV: Survey of Developers," Available at URL: http://www.infonomics.nl/ FLOSS/report/FLOSS_Final4.pdf.

Gruber, M. and J. Henkel (2005) "New Ventures Based on Open Innovation - an Empirical Analysis of Start-up Firms in Embedded Linux," *International Journal of Technology Management* 33 (4), pp. 354-372.

Haefliger, S., G. von Krogh, and S. Spaeth (2008) "Code Reuse in Open Source Software," *Mangement Science* 54 (1), pp. 180-193.

Hair, J.F., Jr., R.L. Tataham, J.E. Anderson, and W. Black (2006) *Multivariate Data Analysis*. Upper Saddle River, NJ: Pearson Prentice Hall.

Hardgrave, B.C., F.D. Davis, and C.K. Riemenschneider (2003) "Investigating Determinants of Software Developers' Intentions to Follow Methodologies," *Journal of Management Information Systems* 20 (1), pp. 123-151.

Hardgrave, B.C. and R.A. Johnson (2003) "Toward an Information Systems Development Acceptance Model: The Case of Object-Oriented Systems Development," *IEEE Transactions on Engineering Management* 50 (3), pp. 322-336

Hars, A. and S. Ou (2002) "Working for Free? Motivations for Participating in Open-Source Projects," *International Journal of Electronic Commerce* 6 (3), pp. 25-39.

Henkel, J. (2006) "Selective Revealing in Open Innovation Processes: The Case of Embedded Linux," *Research Policy* 35 (7), pp. 953-969.

Henkel, J. (2009) "Champions of Revealing - the Role of Open Source Developers in Commercial Firms," *Industrial and Corporate Change* 18 (3), pp. 435-471.

Henkel, J. and C.Y. Baldwin (2009) "*Modularity for Value Appropriation: Drawing the Boundaries of Intellectual Property,"* (March 2009), Working Paper, Harvard Business School.

Hertel, G., S. Niedner, and S. Hermann (2003) "Motivation of Software Developers in the Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* 32 (7), pp. 1159-1177.

Herzberg, F. (1968) "One More Time: How Do You Motivate Employees?," *Harvard Business Review* 46 (1), pp. 53-62.

Isoda, S. (1995) "Experience of a Software Reuse Project," *Journal of Systems and Software* 30, pp. 171-186.

Kim, Y.E. and E.A. Stohr (1998) "Software Reuse: Survey and Research Directions," *Journal of Management Information Systems* 14 (4), pp. 113-147.

Krueger, C.W. (1992) "Software Reuse," *ACM Computer Surveys* 24 (2), pp. 131-183.

Lakhani, K.R. and E. von Hippel (2003) "How Open Source Software Works: "Free" User-to-User Assistance," *Research Policy* 32 (6), pp. 923-943.

Lakhani, K.R. and R.G. Wolf (2005) "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," in Feller, J., B. Fitzgerald, S. Hissam, and K.R. Lakhani (Eds.) *Perspectives on Free and Open Source Software*, Cambridge, MA: MIT Press, pp. 3-22.

Langlois, R.N. (1999) "Scale, Scope, and the Reuse of Knowledge," in Dow, S.C. and P.E. Earl (Eds.) *Economic Organization and Economic Knowledge*, Cheltenham, UK: Edward Elgar, pp. 239-254.

Lee, N.-Y. and C.R. Litecky (1997) "An Empirical Study of Software Reuse with Special Attention to Ada," *Transactions on Software Engineering* 23 (9), pp. 537-549.

Lerner, J. and J. Tirole (2002) "Some Simple Economics of Open Source," *The Journal of Industrial Economics* 50 (2), pp. 197-234.

Majchrak, A., L.P. Cooper, and O.P. Neece (2004) "Knowledge Reuse for Innovation," *Management Science* 50 (2), pp. 174-188.

Mellarkod, V., R. Appan, D.R. Jones, and K. Sherif (2007) "A Multi-Level Analysis of Factors Affecting Software Developers' Intention to Reuse Software Assets: An Empirical Investigation," *Information & Management* 44 (7), pp. 613-625.

Mockus, A. (2007) "Large-Scale Code Reuse in Open Source Software," *1st International Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, MN.

Moore, G.C. and I. Benbasat (1991) "Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation," *Information Systems Research* 2 (3), pp. 192-222.

Morisio, M., M. Ezran, and C. Tully (2002) "Success and Failure Factors in Software Reuse," *IEEE Transactions on Software Engineering* 28 (4), pp. 340-357.

Naur, P. and B. Randell (1968) *Software Engineering; Report on a Conference by the Nato Science Committee*. Brussels, Belgium: NATO Science Affairs Division.

Nunnally, J.C. (1978) *Psychonometric Theory*. New York, NY: McGraw-Hill.

Podsakoff, P.M., S.B. MacKenzie, J. Lee, and N.P. Podsakoff (2003) "Common Method Biases in Behavioral Research: A Critical Review of the Literature and Recommended Remedies," *Journal of Applied Psychology* 88 (5), pp. 879-903.

Ravichandran, T. and M.A. Rothenberger (2003) "Software Reuse Strategies and Component Markets," *Communications of the ACM* 46 (8), pp. 109-114.

Raymond, E.S. (2001) *The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly & Associates 2nd Edition.

Riemenschneider, C.K. and B.C. Hardgrave (2001) "Explaining Software Development Tool Use with the Technology Acceptance Model," *Journal of Computer Information Systems* 41 (4), pp. 1-8.

Riemenschneider, C.K., B.C. Hardgrave, and F.D. Davis (2002) "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models," *IEEE Transactions on Software Engineering* 28 (12), pp. 1135-1145

Roberts, J.A., I. Hann, and S.A. Slaughter (2006) "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management Science* 52 (7), pp. 984-999.

Rossi Lamastra, C. (2009) "Software Innovativeness: A Comparison between Proprietary and Free/Open Source Solutions Offered by Italian SMEs," *R&D Management* 39 (2), pp. 153-169.

Sen, A. (1997) "The Role of Opportunism in the Software Design Reuse Process," *IEEE Transactions of Software Engineering* 23 (7), pp. 418-436.

Sen, R., C. Subramaniam, and M.L. Nelson (2008) "Determinants of the Choice of Open Source Software License," *Journal of Management Information Systems* 25 (3), pp. 207-239.

Sherif, K., R. Appan, and Z. Lin (2006) "Ressources and Incentives for the Adoption of Systematic Software Reuse," *International Journal of Information Management* 26 (1), pp. 70-80.

Spaeth, S., M. Stuermer, S. Haefliger, and G. Von Krogh (2007) "Sampling in Open Source Software Development: The Case for Using the Debian GNU/Linux Distribution," *40th Annual Hawaii International Conference on System Sciences*, Waikoloa, HI.

Stewart, K.J. and S. Gosain (2006) "The Impact of Ideology on Effectiveness in Open Source Software Teams," *MIS Quarterly* 30 (2), pp. 291-314.

Straub, D. (1989) "Validating Instruments in MIS Research," *MIS Quarterly* 13 (2), pp. 147-169.

Subramanyam, R. and M. Xia (2008) "Free/Libre Open Source Software Development in Developing and Developed Countries: A Conceptual Framework with an Exploratory Study," *Decision Support Systems* 46 (1), pp. 173-186.

Tracz, W. (1995) *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*. Reading, MA: Addison-Wesley.

von Krogh, G., S. Spaeth, and S. Haefliger (2005) "Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects," *38th Annual Hawaii International Conference on System Sciences*, Big Island, HI.

von Krogh, G., S. Spaeth, S. Haefliger, and M. Wallin (2008) "*Open Source Software: What We Know (and Do Not Know) About Motives to Contribute,"* (April 2008), Working Paper, DIME Working Papers on Intellectual Property, Available at URL: http://www.dime-eu.org/files/active/0/WP38_vonKroghSpaethHaefligerWallin_IPROSS.pdf.

von Krogh, G., S. Spaeth, and K.R. Lakhani (2003) "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy* 32 (7), pp. 1217-1241.

Watson, S. and K. Hewett (2006) "A Multi-Theoretical Model of Knowledge Transfer in Organizations: Determinants of Knowledge Contribution and Knowledge Reuse," *Journal of Management Studies* 43 (2), pp. 141-173.

West, J. (2003) "How Open Is Open Enough? Melding Proprietary and Open Source Platform Strategies," *Research Policy* 32 (7), pp. 1259-1285.

Wu, C.-G., J.H. Gerlach, and C.E. Young (2007) "An Empirical Analysis of Open Source Software Developers' Motivations and Continuance Intentions," *Information & Management* 44 (3), pp. 253-262.

Ye, Y. and G. Fischer (2005) "Reuse-Conducive Development Environments," *Automated Software Engineering* 12 (2), pp. 199-235.

Appendix

Table A1: Factor Analysis and Reliability of Developer Motivation Constructs

	Rotated component matrix						
Construct/item	1	2	3	4	5	6	Cronbach's α
1. Challenge seeking							0.807
Chal1	0.052	**0.794**	0.137	0.203	0.007	0.043	
Chal2	-0.031	**0.891**	0.119	0.135	0.034	0.019	
Chal3	0.020	**0.794**	0.075	0.172	-0.026	0.026	
2. Coding fun and enjoyment							0.746
Fun1	0.021	0.176	0.122	**0.763**	-0.024	0.111	
Fun2	-0.008	0.284	0.217	**0.718**	0.100	0.005	
Fun3	0.038	0.165	0.077	**0.839**	0.010	0.002	
3. Community commitment							0.640
Com1	-0.068	0.043	0.109	0.055	0.154	**0.743**	
Com2	0.138	0.112	0.010	0.027	-0.099	**0.691**	
Com3	-0.051	-0.017	0.089	0.033	0.186	**0.832**	
4. Skill improvement							0.758
Learn1	0.101	0.148	**0.832**	0.162	0.003	0.044	
Learn2	0.192	0.120	**0.831**	0.159	0.027	0.058	
Learn3	0.034	0.093	**0.721**	-0.005	0.190	0.125	
5. OSS reputation building							0.901
OSSRep1	0.253	-0.004	0.053	0.035	**0.892**	0.098	
OSSRep2	0.240	0.021	0.055	0.010	**0.900**	0.091	
6. Commercial signaling							0.866
ComSig1	**0.847**	0.004	0.178	0.065	0.095	0.019	
ComSig2	**0.857**	-0.027	0.087	-0.007	0.250	-0.016	
ComSig3	**0.800**	0.056	0.045	-0.009	0.359	-0.031	

Notes: The factor analysis uses principal component analysis and Varimax rotation; high factor loadings under each component in the rotated matrix are indicated by bold text and gray shading.
N=624.

Table A2: Discriminant Analysis of Developer Motivation Constructs

	1	2	3	4	5	6
1. Challenge seeking	**0.757**					
2. Coding fun and enjoyment	0.444***	**0.705**				
3. Community commitment	0.112***	0.132***	**0.657**			
4. Skill improvement	0.285***	0.323***	0.207***	**0.751**		
5. OSS reputation building	0.033	0.064	0.194***	0.189***	**0.906**	
6. Commercial signaling	0.047	0.063	0.026	0.254***	0.495***	**0.832**

Notes: The diagonal bolded entries are square roots of the average variance extracted (AVE) of the respective construct; the off-diagonal entries are standardized correlations between constructs; * correlation significant at 10%; ** correlation significant at 5%; *** correlation significant at 1% level.
N=624.

Table A3: Exploratory Factor Analysis of Reuse Benefits				
	Rotated component matrix			
Item (Rank in Figure 2)	1	2	3	4
Difficult Problem (Rank 3)	0.081	0.171	0.090	**0.948**
Faster (Rank 1)	0.181	**0.793**	-0.001	0.326
Most Important (Rank 2)	0.176	**0.834**	0.236	0.062
Most Fun (Rank 6)	-0.021	0.414	**0.743**	0.021
Outs Maintenance (Rank 7)	0.332	-0.029	**0.779**	0.162
Reliable SW (Rank 4)	**0.840**	0.278	0.130	-0.031
Secure SW (Rank 8)	**0.872**	0.124	0.113	0.090
Standard SW (Rank 5)	**0.739**	0.002	0.097	0.237
Notes: The factor analysis uses principal component analysis and Varimax rotation; high factor loadings under each component in the rotated matrix are indicated by bold text and gray shading. N=624.				

Table A4: Exploratory Factor Analysis of Reuse Issues and Drawbacks			
	Rotated component matrix		
Item (Rank in Figure 3)	1	2	3
Finding (Rank 9)	**0.854**	0.089	0.036
Understanding (Rank 7)	**0.876**	0.125	0.073
Adapting (Rank 6)	**0.847**	0.165	0.087
Quality Risks (Rank 5)	0.156	**0.934**	0.100
Security Risks (Rank 4)	0.088	**0.935**	0.084
Performance Loss (Rank 8)	0.231	**0.451***	0.284
Installation (Rank 2)	0.152	0.089	**0.764**
Dependence (Rank 1)	-0.051	0.118	**0.785**
Additional Work (Rank 3)	0.162	0.162	**0.707**
Notes: The factor analysis uses principal component analysis and Varimax rotation; high factor loadings under each component in the rotated matrix are indicated by bold text and gray shading. *The loading of this item on its construct is rather low, however, it is retained due to the good overall Cronbach's α of the construct (0.76). N=624.			

Table A5: Descriptive Statistics of Explanatory Variables Used in Table 6

Variable	Dummy variable equal to "1" if…	Frequency of "0"	Frequency of "1"
ProjPolSupport	Developer's current main project has a policy encouraging its developers to reuse	438 (70%)	186 (30%)
ProjPolDiscourage	Developer's current main project has a policy discouraging its developers from reuse	606 (97%)	18 (3%)
ProjStandalone	Developer's current main project is a standalone executable application project and not a component project	162 (26%)	462 (74%)
DevProf	Developer is working as professional developer or has worked as professional developer for a firm	191 (31%)	433 (69%)
DevEduReuse	Developer has received training on reuse during her education	412 (66%)	212 (34%)
DevProfEduReuse	Developer has received training on reuse when working as software developer for a firm	544 (87%)	80 (13%)
Residence-N.America	Developer resides in North America	455 (73%)	169 (27%)
Residence-S.America	Developer resides in South America	594 (95%)	30 (5%)
Residence-Asia&RoW	Developer resides Asia, Africa, Australia or Oceania	536 (86%)	88 (14%)

Variable	Explanation	Min.	Max.	Med.	Mean	S.D.
Benefit-Effectiveness	Factor score from exploratory factor analysis… on developer's perception of effectiveness effects of code reuse	-4.762	2.047	0.178	0	1
Benefit-Efficiency	…on developer's perception of efficiency effects of code reuse	-3.568	2.313	0.093	0	1
BenefitQuality	…on developer's perception of quality effects of code reuse	-3.972	2.909	-0.027	0	1
Benefit-TaskSelection	…on developer's perception of task selection effects of code reuse	-3.884	3.026	0.033	0	1
Issue-ControlLoss	…on developer's perception of control loss effects of code reuse	-3.781	2.376	0.065	0	1
DevOSS-Netsize (log)	Size of developer's personal OSS network (as logarithm)	0	6.217	2.197	2.001	1.033
DevOthe-rProjects	Number of OSS projects besides current main project, that developer has ever been involved in	0	48	2	3.617	5.388
ProjPhase	Development phase of developer's current main project (1=Pre-Alpha, 2=Alpha, 3=Beta, 4=Stable/ Production, 5=Mature)	1	5	3	3.221	1.184
MotChallenge	Index variable constructed from challenge scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	5.333	5.128	1.060
MotFun	Index variable constructed from fun scale (1=Strongly disagree,…, 7=Strongly agree)	1.667	7	5.000	5.152	1.092
MotLearning	Index variable constructed from learning scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	5.333	5.317	1.100

Mot-Community	Index variable constructed from community commitment scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	5.667	5.614	1.003
MotOSS-Reputation	Index variable constructed from OSS reputation scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	4.000	3.609	1.621
MotSignaling	Index variable constructed from signaling scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	4.667	4.312	1.527
DevNorm	Index variable constructed from subjective norms scale (1=Strongly disagree,…, 7=Strongly agree)	1	7	4.000	3.927	1.555
ConditionLack	Developer's agreement (1=Strongly disagree,…, 7=Strongly agree) to… lack of reusable code as impediment to reuse	1	7	4	3.784	1.823
Condition-License	… issues with license incompatibilities as impediment to reuse	1	7	2	3.006	1.852
Condition-Language	… issues with programming language incompatibilities as impediment to reuse	1	7	2	2.154	1.401
Condition-Architecture	... issues with project architecture as impediment to reuse	1	7	2	2.630	1.597
DevSkill	Self-assessment of developer's software development skills compared to the average OSS developer (1=Much worse,…, 5=Much better)	1	5	3	3.269	0.989
ProjSize	Size of developer's current main project in number of developers	1	999*	2	6.091	44.420
Proj-Complexity	Complexity of developer's current main project compared to average project on SourceForge.net (1=Much less complex,…, 5=More more complex)	1	5	3	2.947	1.029
ProjStack	Position of developer's current main project in software stack (1=Very low,…, 5=Very high)	1	5	4	3.333	0.921
DevOSS-Experience	Number of years developer has been active working on OSS projects	1	40**	5	5.668	4.709
DevProjTime	Average weekly hours developer works on her current main project	0.5	58	5	8.775	10.723
DevProjShare	Share of work that has been done by developer in her current main project as opposed to other project team members	5	100	90	67.436	36.998

*The main project of this developer is Linux where a very high number of project team members seems reasonable.
**This developer claims to have been involved in OSS even before it got started. We assume that she implies that she has already been working on a project that later became OSS at that point in time.
N=624.

Table A6: Correlation Matrix of Explanatory Variables Used in Table 6 – Continued on Next Page																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1. BenefitEffectiveness	1.00																	
2. BenefitEfficiency	n.m.	1.00																
3. BenefitQuality	n.m.	n.m.	1.00															
4. BenefitTaskSelection	n.m.	n.m.	n.m.	1.00														
5. IssueControlLoss	n.m.	n.m.	n.m.	n.m.	1.00													
6. DevOSSNetsize		0.14	0.11			1.00												
7. DevOtherProjects	-0.08					0.31	1.00											
8. ProjPhase			0.07	-0.10		0.17	0.17	1.00										
9. MotChallenge								-0.08	1.00									
10. MotFun				0.08			0.09	-0.08	0.44	1.00								
11. MotLearning	0.08			0.16	-0.09			-0.12	0.29	0.32	1.00							
12. MotCommunity	0.09	0.14	0.14		-0.07	0.22	0.13	0.10	0.11	0.13	0.21	1.00						
13. MotOSSReputation			0.15	0.10	-0.08	0.13	0.15	0.09			0.19	0.19	1.00					
14. MotSignaling			0.07	0.16				-0.10			0.25		0.50	1.00				
15. DevNorm	0.07	0.19	0.26	0.21					0.09	0.07	0.10	0.12	0.18	0.12	1.00			
16. DevSkill		0.12	0.07		0.13	0.10	0.16	0.15	0.09		-0.15	-0.12	0.12	0.11		1.00		
17. ProjPolSupport		0.10	0.09			0.23	0.19	0.15	0.10		0.09	0.19	0.11		0.12	0.12	1.00	
18. ProjPolDiscourage		-0.09			0.12	-0.07				-0.08		-0.07				0.09	-0.11	1.00
19. ConditionLack	-0.08	-0.19		-0.08												-0.08	-0.09	
20. ConditionLicense	-0.10	-0.15					0.11					-0.13	0.12	0.13			0.18	
21. ConditionLanguage		-0.23			0.08											-0.12		
22. ConditionArchitecture		-0.16	-0.08		0.07				-0.07					-0.14		-0.12		
23. ProjSize			-0.07		-0.08		0.19	0.11					0.09	0.09		0.08	0.11	
24. ProjComplexity	-0.11		0.12		0.09	0.18	0.19	0.21					0.09	0.11		0.38	0.30	
25. ProjStack		0.15										0.11	-0.07	-0.07		-0.11		
26. ProjStandalone										0.07		0.08	-0.17	-0.16		-0.14		-0.09
27. DevOSSExperience			0.13	-0.08		0.26	0.29	0.29			-0.15	0.12		-0.13		0.25	0.09	
28. DevProjTime	-0.09					0.13	0.13		0.11	0.12	0.11		0.10	0.21	0.09	0.17	0.29	
29. DevProjShare						-0.21	-0.08	-0.22		0.07							-0.43	-0.10
30. DevEduReuse					-0.09			-0.11				-0.09				0.12		
31. DevProfEduReuse		0.08				0.08						-0.10				0.12		
32. DevProf		0.13	0.11			0.08		0.08		-0.08	-0.10	-0.14	0.07	0.15	0.08	0.39	0.07	
33. Residence-N. America		0.07				-0.07												
34. Residence-S. America						0.07					0.07							
35. Residence-Asia & RoW		-0.08						-0.09				-0.07						

Table A6: Correlation Matrix of Explanatory Variables Used in Table 6 – Continuation from Previous Page																	
	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
18. ProjPolDiscourage																	
19. ConditionLack	1.00																
20. ConditionLicense	0.17	1.00															
21. ConditionLanguage	0.23	0.24	1.00														
22. ConditionArchitecture	0.24	0.11	0.33	1.00													
23. ProjSize		0.08			1.00												
24. ProjComplexity	-0.09	0.18		-0.09	0.16	1.00											
25. ProjStack		-0.13	-0.07	0.07	-0.11		1.00										
26. ProjStandalone		-0.101		0.		0.13	0.37	1.00									
27. DevOSSExperience				0.13	0.07	0.23	-0.08		1.00								
28. DevProjTime	-0.08	0.14			0.15	0.38		0.11		1.00							
29. DevProjShare		-0.20	-0.08	-0.08	-0.17	-0.34			-0.10	-0.15	1.00						
30. DevEduReuse									-0.07			1.00					
31. DevProfEduReuse					0.09							0.08	1.00				
32. DevProf			-0.12	-0.08		0.08	-0.09	-0.14	0.09			0.18	0.25	1.00			
33. Residence-N. America									0.15		0.09				1.00		
34. Residence-S. America									-0.08			0.08			n.m.	1.00	
35. Residence-Asia & RoW															n.m.	n.m.	1.00
Notes: Only correlations with p<0.1 are shown; n.m. = not meaningful because variables are dummy variables coding the same characteristic or are scores of the same exploratory factor analysis																	

Table A7: Multivariate Analysis of Developers' Reuse Behavior – Robustness Check			
	Past importance of reuse		(6) Future importance of reuse (Likert scale)
	(4) Likert scale	(5) Percentage scale	
Attitude toward reuse			
BenefitEffectiveness *(H1a)*	0.220*** (0.076)	2.464** (1.010)	0.146** (0.062)
BenefitEfficiency *(H1b)*	0.634*** (0.080)	6.047*** (1.059)	0.499*** (0.066)
BenefitQuality *(H1c)*	0.322*** (0.079)	2.262** (1.048)	0.273*** (0.065)
BenefitTaskSelection *(H1d)*	0.157** (0.077)	3.368*** (1.026)	0.144** (0.064)
IssueControlLoss *(H1e)*			
Access to local search			
DevOSSNetsize (log) *(H2a)*	0.172** (0.080)	2.307** (1.047)	0.246*** (0.066)
DevOtherProjects *(H2b)*	0.030* (0.015)	0.465** (0.196)	0.034*** (0.013)
Project maturity			
ProjPhase *(H3)*	-0.124* (0.066)	-2.984*** (0.871)	-0.204*** (0.054)
Compatibility with project goals			
MotChallenge *(H4a)*		-2.466** (0.962)	
MotFun *(H4b)*			
MotLearning *(H4c)*			
MotCommunity *(H4d)*	0.180** (0.081)	1.912* (1.067)	0.163** (0.066)
MotOSSReputation *(H4e)*			
MotSignaling *(H4f)*			
Subjective norms			
DevNorm	0.120* (0.065)	2.133** (0.870)	0.205*** (0.054)
Perceived behavioral control			
ProjPolSupport	0.405** (0.180)		0.335** (0.143)
ProjPolDiscourage	-1.210*** (0.447)		-1.299*** (0.375)
ConditionLack	-0.236*** (0.042)	-2.355*** (0.564)	-0.160*** (0.035)
ConditionLicense			
ConditionLanguage			
ConditionArchitecture			
DevSkill			
Further control variables			
ProjSize			
ProjComplexity			
ProjStack	0.232*** (0.083)		0.172** (0.069)
ProjStandalone			
DevOSSExperience			
DevProjTime	0.016** (0.007)		
DevProjShare			
DevProf			
DevEduReuse			
DevProfEduReuse	0.573** (0.232)	5.581* (3.012)	0.414** (0.189)
Residence-N. America			
Residence-S. America			
Residence-Asia & RoW			
Constant	3.145*** (0.622)	34.228*** (8.393)	2.858*** (0.509)
Observations	624	624	624
Pseudo R^2	0.101	0.026	0.112
Likelihood ratio	$X^2(15)=252.81$, p<0.0001	$X^2(12)=149.36$, p<0.0001	$X^2(14)=272.67$, p<0.0001
σ	1.814	24.600	1.514

Notes: All models are Tobit models; standard errors in parentheses; * significant at 10%; ** significant at 5%; *** significant at 1%. Eliminated variables are also jointly insignificant.