2014

Performance Implications of Stage-Wise Lead User Participation in Software Development Problem Solving

Jorge A. Colazo
Trinity University, jcolazo@trinity.edu

Performance implications of stage-wise lead user participation in software development problem solving

Jorge Colazo

Dept. of Finance and Decision Sciences

Trinity University

jcolazo@trinity.edu

Forthcoming at *Decision Support Systems. Final version submitted to DSS August 2014*

Abstract

The problem-solving view of new product development sees the innovation process as a series of problem-solving loops broken down into three stages: problem detection, analysis and removal. We link this framework with lead user-driven innovation regarding software and show that effort by lead users (LUs) in each stage of the innovation problem solving process is, in varying degrees, associated with the source code's quality, the productivity of the development process and the software's popularity. We also test whether front loading the problem solving process is associated with development performance and we find that front loading is associated with increased code quality but decreased development productivity. Empirical tests are carried out with data from open source software projects. Findings potentially impact the design and management of online communities to help product development.

1 Introduction

Despite the obvious economic importance of software projects, a great number of new software development projects cannot be considered successful, with end products plagued by quality issues, not to mention projects that are routinely late or extremely over budget. While estimates of failure rates range between 15% and 70% [36, 84] depending on the kind of software project and how success is measured, it is generally accepted that the economic and social consequences of unsuccessful software development projects warrant continued attention from practitioners and academia [44].

Various development management methodologies have been advanced to alleviate new software development problems, for instance agile or lean development, with varying rates of success [25]. A relatively recent trend in innovation can, however, complement most project management techniques: the concept of user aided innovation.

User-aided innovation –the development of new products with active help from users- was placed in the research agenda by Von Hippel [94], followed up by a relatively recent stream of studies [39, 50, 61, 97]. These works particularly highlight users who experience product related needs well ahead of the mainstream consumer and stand to benefit significantly from product modifications, so much that they often carry those modifications out themselves. These users are called "lead users" (LUs) [94].

LU involvement in product development has been documented in areas as diverse as industrial equipment and extreme sports gear [77]. The software industry can be considered a pioneer in exploiting this resource. For instance, long ago before the concept of the LU was coined, it became standard practice in the software industry to get select user feedback by limitedly releasing “beta” versions. Software manufacturers, especially in the gaming industry, routinely facilitate tools that enable advanced users to

produce modifications or “mods”, some of which are eventually incorporated into the official version of a product [75]. Early studies on LU innovation were carried out looking at industrial control systems [92], and open source software has been repeatedly cited as a prototypical case of LU involvement [10, 43, 99]. Some proprietary software companies have experimented with the concept of LU involvement to aid their product development processes, like Microsoft did with its Most Valuable Professionals (MVP) program [70] or Dell with its Idea Storm community [24].

Through online communities, software LUs can provide ideas for features [99] or inform developers of defects in a product and discuss alternative strategies to solve these problems, even suggesting specific solutions. It is interesting to note that LU involvement in software development is related mainly to defect reporting, analysis and repair [18, 31, 56]. These activities closely match the three main stages in the innovation problem solving process: problem detection, analysis and removal [47, 49].

User-aided innovation’s advantages and problems have mostly been studied in company-led settings [33, 45, 63]. However, alongside potential benefits, involving LUs in the software development process through online LU communities poses several particular challenges. First, participation in online user communities is mostly voluntary, and motivating LUs to participate in the development process is a prime concern [7]. Second, user-aided problem solving activities can occur simultaneously; i.e. more than a single defect can be worked on by LUs at any given time, and LUs usually prioritize their work according to their own desires and abilities rather than by a formal development blueprint or management dictated priorities, requiring some degree of coordination. Furthermore, LUs can choose to participate in any one or more of the stages of the problem solving process with different intensities and timings.

Project managers may need to spend substantial resources in order to coordinate and incorporate user input from the open source community [20]; contributions need to be screened for their value, feasibility

and appropriateness to the project's technical and commercial objectives, and while part of this assessment can also be done with the LUs' help, the management team eventually decides whether to consider a detected defect, accept the proposed avenue to arrive at a solution or accept a specific solution proposal. Given that resources are limited, a reasonable research question of practical importance is: *What are the performance implications of encouraging LUs to work on problem detection, analysis or removal?* The innovation literature consistently refers to problem solving with regard to the development of a new product as a one-step action, but informed by the problem solving literature we can separately test problem detection, analysis and removal to observe a more nuanced picture.

Furthermore, managing LU contributions in the different stages of problem solving entails varying kinds of resources use as well as benefits, suggesting the potential existence of tradeoffs among the different dimensions of innovation success such as development productivity, code quality or development timeliness. For instance, while it is plausible to suggest that if the community has more LUs detecting defects we may eventually arrive at better code quality, the processing of LUs' input can hinder the development team's productivity. Another research question is: *Are there any performance tradeoffs involved in different LU participation intensities along the stages of the problem solving process?* Since innovation success is multidimensional, empirically testing several kinds of outcomes produces a more complete understanding of problem solving vs. innovation performance.

Also, timing in problem solving is important: Thomke and Fujimoto [90] looked at a few projects in the automobile industry and observed improvements in some product development performance metrics when they attempted to front load their product development process; i.e. favored early (rather than late) identification and solution of problems. Their study relied on handpicked projects and anecdotal evidence, in an environment devoid of LU participation. To date, empirical studies based on large samples and hard metrics about of the potential benefits of front loading in the context of LU participation are lacking. We

attempt to fill such gap with this study. Our third research question is: *Is development performance associated with the relative promptness of LU participation in problem solving?*

Answering these questions is useful to design online communities and allocate resources to encourage or discourage user participation in the different stages of problem solving. For instance, a manager may wish to only get user feedback about defects but not engage LUs in problem resolution, while another may want to post in-house detected bugs for analysis by volunteers, depending on costs and desired outcomes.

This study draws from the problem solving and LU innovation literature to devise hypotheses that are tested using ordinary least squares models built from data extracted from software development projects. After this Introduction, Section 2 of the paper includes a review of the literature, leading to the hypotheses to be empirically tested. Section 3 explains the research setting, while analytical methods are developed in Section 4. Section 5 presents the results of the analyses, while Section 6 offers a discussion of the findings and Section 7 reviews the limitations of this study and its implications for future research.

2 Literature Review and Hypotheses

Users have been purposely and routinely involved at the market research stage of new product development in order to identify their needs, and help design or refine product offerings. For example, the use of focus groups is a widely known such practice [85]. Lately, the involvement of users has shifted to a more active role.

We know that innovations frequently originate from users' initiatives [27]. More recently, [93, 94] it has been observed that a subgroup of users not only generates ideas for improvements to existing products but also they carry out modifications themselves, to address specific needs. These special users have been dubbed lead users or "LUs". Two independent dimensions [33] are related to LUs: they 1) experience

product-related needs well ahead of the mainstream consumers and 2) stand to significantly benefit from product modifications.

One stream of research focused on characteristics of LUs. They seem to have, compared to mainstream users, a longer use experience [78]. Many times, dissatisfaction with the current product features seems to trigger product modifications, and LUs possess some particular personality traits such as heightened locus of control and increased innovativeness [79]. LUs also display more willingness to collaborate, better product related functional knowledge and a high level of strategic alignment with a brand's identity [63, 76]. They also tend to be opinion leaders rather than opinion seekers [77].

Another area of research looks at the consequences of LU innovation. LUs are sought not only because of the economic value of their suggestions, but because their needs predate those of the general market and solutions to problems they point out often can be transferred into other components of the development portfolio [94]. Innovations spawned by LUs are more likely to be breakthrough innovations and to produce relatively higher profit margins [26, 59]. Products based on LU-generated ideas are more likely to become commercially attractive [92], and they are more original, but less feasible [55].

Though the performance of products under the LU innovation paradigm has been extensively researched, the performance of LU-driven development processes has received relatively little attention [32]. Previous studies suggested that LU involvement in idea generation could impact productivity or quality [92, 94], but little empirical support is available, which this study intends to alleviate.

LUs can offer their contributions at the sponsoring firm's premises, as it is done by 3M [60], but the Internet has given birth to firms' use of online product user communities to generate LU ecosystems. In these virtual spaces, by means of e-mail list servers, Internet relay chat rooms and bulletin boards, LUs

can get in contact with each other and with personnel from the sponsoring firm. Product performance is discussed, defects reported, and modifications and solutions suggested to known problems. The relative lower cost of accessing LUs by this means vis-à-vis by face to face interaction allows online communities to be larger, remain active for longer proportions of the product development life cycle, and establish relatively stable memberships, which tends to be effective at preserving organizational learning [51].

Whether company sponsored or not, online product user communities exist for a wide range of products, from sporting equipment to music instruments to software [1, 45]. In the software area, giants of proprietary hardware and software such as Dell [24] and Microsoft [70] invested substantive resources to tap into communities of LUs.

Outside of the proprietary software realm, open source software (OSS) projects have been repeatedly considered in studies about LU innovation [32, 57, 96]. OSS is software developed under licensing terms that make publicly available the complete source code of the product and allow the redistribution of modified versions. A prototype of the program is posted in a publicly viewable electronic repository and afterwards LUs contribute their work voluntarily, at varying skill levels, making intensive use of electronically mediated communication. OSS development is highly flexible and allows for the quick rearrangement of labor to adjust to changing priorities. The involvement of LUs in OSS development has been well documented and is considered key to its thriving [81, 95].

The work carried out by LUs can be inscribed into the problem solving view of new product development [21, 30]. This paradigm considers innovation as a bundle of iterative problem solving activities [65, 89, 90]. The activities involved in product design improvement can be described as following a three step cycle where the main stages are problem detection, analysis and removal [89]. With software, on line LU communities clearly engage in problem solving in those three stages. Bug reports signal the problem

detection stage; feedback generated by the developers' mailing list reflects problem analysis; and the submission of patches represents the problem removal stage.

The first set of hypotheses refers to problem detection vs. performance. Problem detection effort is the effort put in place by lead users when they come across a problem, realize it is a software bug and decide to report it. New product development (innovation) performance has been characterized in terms of both product success and development project success [54]. The former dimension can be measured by product quality, product market share (popularity, or user base), or product sales, and the latter by project cost, development speed, timeliness and development productivity [9, 46]. Productivity and time-based measures of project success were recommended for high-tech products, such as software [6]. We wanted to cover both dimensions of success (product and process) with the available data, and hence performance is going to be analyzed in terms of software quality and software popularity for the product-related dimension and software development productivity for the process-related dimension. Of several possible quality metrics for software, we are going to consider, due to the availability of data, the number of pre-test defects expected in the software [69].

In the problem detection stage, a large group of lead users explore the product's functionality in different ways, placing the product under working conditions that may not have been contemplated in the original test cases. There is a stark difference on how a proprietary software company tests for bugs and user testing. The former relies heavily on "test cases" [68] which are especially designed processing jobs that test known or highly impactful critical fail points looking for particular fail events that are normally linked to the formal design requirements. These test cases are very useful indeed, because they assure that the software will do what is supposed to do, not catastrophically failing in the attempt. We can say that formal testing is proactive and relatively limited, i.e. the software is proactively fed a pre-designed test

case and if software crashes, a bug is recorded and later investigated. Bugs that are not supposed to be caught with the formal test cases are of course not going to be detected.

User testing, on the contrary, is reactive and relatively comprehensive. Users do not run formal test cases and do not look for specific fail modes. Most times they are not even trying to test the software, but rather they fortuitously run into a problem, prompting them to submit a bug report. User testing is arguably (or at least can potentially be) much more comprehensive than formal case tests because user testing will try every possible expected use and many unexpected or previously unconsidered functionalities of the software, which is particularly true of LUs, since they operate at the edge of the product's possibilities. In brief, formal testing is limited, and strongly related to design requirements. User testing is (potentially) unlimited, and not as strongly linked to original design requirement.

Then, H1a posits that the more the expected bugs the more detection effort will be observed, simply because when they are canvassing the software, users will run into more bugs (and report them) than otherwise if the software had fewer expected bugs. Along the reverse association path, the more effort is observed in bug detection (for instance more bug reports) then it will likely be in response to a higher level of expected defects. Then, there is a direct association between the number of expected bugs and bug reports, or an inverse relationship between expected quality and bug reporting activity.

Although the detection of defects is largely an activity that LUs can undertake independently of the core developers, bug reports need to be studied and understood by those LUs who can code. Processing and analyzing defect reports takes considerable effort because one must match a problem with specific lines of code, rooting out the origin of the problem [24] before writing new lines of code. The time used to fix problems distracts from proper development activities that will grow the functionality of the code base. While fixing a defect can add or delete lines of code, adding functionality generally grows the code base

significantly because new algorithms and control structures need to be put in place. Then we can expect a negative association between number of LUs that analyze problems and development productivity. Finally, a larger user base will include more LUs that may find problems, and we can expect that more popular products are going to have more defects detected. All in all:

H1a: Problem detection effort by LUs is inversely related to expected code quality.

H1b: Problem detection effort by LUs is inversely related to productivity.

H1c: Problem detection effort by LUs is directly related to product popularity.

Separate from the problem detection step, where LUs work primarily independently on testing the product and reporting defects, the analysis of alternatives may not only be individual but can be a group effort, with LUs interacting through ad-hoc mailing lists and chat rooms criticizing or supporting others' suggestions, ideally arriving at one alternative that is sufficiently palatable for the community, feasible and in accordance with the project's goals [101, 102].

In particular, collaborative problem analysis is at the core of what defines the open source software development model, under the premise that "Given enough eyeballs, all bugs are shallow" [73], meaning that crowdsourcing bug analysis enables, by task partitioning and by including a broader set of skills, the efficient solution of even very complex defects, which for a single developer would be a daunting task.

At the problem analysis stage, we define problem analysis effort as the effort put in place to understand and offer a potential solution for every bug detected. The first hypothesis in this set refers to the relationship between expected code quality and problem analysis effort.

Since even in large LU communities resources such as time and access to technology are still limited, we can expect that the bigger the number of expected defects, a relative smaller portion of those resources

can be assigned to analyze each problem and the effort per problem is going to be smaller. On the reverse reasoning, the less effort spent in analyzing problems, the more defects will remain, and then:

H2a: Problem analysis effort by LUs is directly related to expected code quality.

Discussions to decide the best way to reach a solution require some degree of consensus building and the coordination of dissimilar points of view. The problem solving literature suggests that consensus takes more effort in larger groups than in smaller groups, which has been observed in online communities [35]. Effort in coordination is not value-added and does not contribute to core productivity; since resources are limited, the higher the degree of collaborative analysis the bigger the need for coordination and the proportion of value-added time for growing the code base will be smaller and net productivity will suffer.

This effort to arrive at the best alternative for solving the problem through more intense analysis will also place an administrative overhead, potentially overloading the capacity of administrators to manage the size of the solution sample space. In other words, more analysis requires more coordination at both the LU level and the administrators' level, with both effects detrimental to productivity, because they decrease the available time to grow the code base's functionality, and then:

H2b: Problem analysis effort by LUs is inversely related to development productivity.

Relevant literature informs us that quality of problem solving generally has a direct association with the size of the group attempting its solution [100]. This is based on individuals having limited cognitive faculties, and selecting partially optimum solutions constrained by previous experience. Those partial solutions are augmented and improved by collaboration with members with different worldviews and skill sets [2]. The size of a group (in this case those who analyze problems) is also related to its diversity [2], and in turn diversity was found to be directly related to problem solving quality and creativity [34, 40]. Crowdsourcing for collaborative problem solving has been shown to generate significantly more creative solutions, with those solutions not posing additional problems from the feasibility point of view [71].

Basically, the more different points of view are constructively brought into the debate, the more thorough the final solution can be, and also the fewer new defects will likely be introduced as part of the solution:

H2c: Problem analysis effort by LUs is directly related to product popularity.

Finally, in the problem removal stage once a course of action has been set and the best alternative solution chosen, the fix is implemented. Since solutions address found problems, it could be expected that the higher the number of expected bugs the more LUs will submit solutions.

H3a: Problem removal effort by LUs is inversely related to expected code quality.

Solutions suggested by LUs need to be scrutinized by core team members before incorporating them into product, and core team members will use their time to review contributions instead of progressing with the product's functionality. Fixing an error does not necessarily involve a substantial growth of the code base; many times it requires deleting instructions. On the contrary, adding functionality or complexity to the program involves superlinear code base growth [37] whereas changes for fixes tend to be much smaller [72]. However, in terms of time invested, reviewing others' solutions to a problem is often difficult, and the time demanded can be substantial [98]. Then, an inverse association will be hypothesized.

H3b: Problem removal effort by LUs is inversely related to development productivity.

LUs oftentimes act as evangelists for products, actively disseminating their qualities and causing other people to be aware of and use the product [87]. The more popular the product, the more LUs there will be, exerting stronger effort to remove defects while promoting it to the general public. Moreover, if there is a large group of LUs who solve quality problems, mainstream users also will be less frustrated by product glitches and will tend to use more of the product and recommend it to peers:

H3c: Problem removal effort by LUs is directly related to product popularity.

Front loading the development process seems to have originated formally in the automotive industry [90], where evidence from development projects supports that the earlier problems are fixed the higher the quality of the final product and the fewer post introduction defects. The Toyota Product Development

System [67], suggests that in innovation, most resources should be concentrated in the analysis of different points of views and in the careful convergence of those points of view prior to prototyping. This process includes working out potential pitfalls. Agile and lean software development methodologies, drawing from the lean manufacturing literature, encourage early and rapid feedback with clients and among developers, with the aim of reducing both defects and maintaining code down the way [62]. Although the effect of front loading the problem solving process has not to our knowledge been empirically tested on a large sample, it is common in software development to delay defect removal to the back end of development, generating increased instability as the defective part is interfaced with further functionality [3]. Similarly, defects left for late solution are more difficult to understand and their solutions more elaborate, costlier and more exhaustive to fix, and requiring more hours' work [91] . If not fixed early, software bugs can infect new code and create new bugs, which are more difficult to find and fix since their fixing involves more time and even more lines of code [3]. Working on a more complex product is also more difficult and error prone. Then, assuming there is no uneven skills distribution along the development timeline, we can expect that if relatively more defects are fixed earlier, there will be fewer latent defects left in the code, and the additional burden of extra labor needed for late defect removal can be avoided:

H4a: Front loading of problem removal by LUs is directly related to code quality

H4b: Front loading of problem removal by LUs is directly related to productivity

3 Research Setting

A potential research setting for this empirical study should include a user community that has been identified as routinely taking advantage of active participation from LUs in problem solving activities. It also should yield such availability of data that can allow the calculation of the extent of LU involvement in each of the three different stages of the problem solving cycle (problem detection, analysis and

removal) separately, as well as suitable performance metrics. With these conditions in mind, the research setting chosen for this study comprised open source software development projects.

OSS projects include the active participation of LUs. The general structure of OSS project teams has been described previously [17]. There is a subgroup or core that stands out from the rest in terms of intensity of work [58]. The core is made up of developers ("core developers") who hold rights to review code submissions and write modifications into the Source Code Control System (SCCS) of the project, an online source code repository that prevents code conflicts and allows for coordinated collaboration. The core developers decide which contributions will be part of the source or be rejected or modified for acceptance. Core developers are "project owners" and they will not be considered part of the LU group.

Outside the core there is a larger group, co-developers, who contribute to the project according to their skills. Some contribute with new source code while others contribute code reviews or bug fixes [17, 29, 66] in the form of patches [66] . The number of co-developers is generally larger (often in orders of magnitude) than the number of core developers.

In addition to this generally accepted subdivision of developers into core and peripheral groups, a structural layer that is further removed from the main coding task is composed of users. Though most users are passive, just using the software without providing specific feedback, a subgroup of users is called "active users" and they submit bug reports, feature requests and software patches. These members normally do not have a level of technical expertise as high as those of the core and co-developers, but are able to test the product and report defects. Active users have the characteristics of LUs [33, 94] and their activity is evident in the project's bug tracking system.

In brief, the OSS project communities are characterized by a tight, smaller group of core developers that have a final say in the acceptance of contributions from the user base. The different layers can be identified by looking at various community artifacts such as SCCS logs and bug trackers.

The different artifacts visible in OSS development can be used to gather information about the three stages in the problem solving cycle. Defects in software are identified by LUs, who submit a bug report with information that allows developers to reproduce the problem in order to find the cause and devise solutions. Once a bug is discovered, its description is posted to the public and developers discuss possible courses of action through the developers' mailing list. Eventually, they select one course of action that will be coded into a patch, which is sent to the core group for addition to the source code base.

Although software defects can be found and fixed at the design stage, we are concerned in this paper with those defects found and fixed after the first working prototype is released. Since it can be argued that that software is in a constant product development process and we only have access to substantial information once a product is released, we believe the approach is valid and practical. Further, most available data refers to coding activity and bug fixing after there is a working version of the product.

4 Methods

This section describes sampling, data collection, measurement and analysis. The hypotheses are tested using ordinary least squares (OLS) regression.

4.1 Sampling

The population of interest is those OSS projects of medium to large developer membership. A majority of OSS projects are individual and many never progressed to the coding stage [53]. In empirical OSS studies we need to both consider projects of practical interest and cull an appropriate sample size. For instance, a

study on OSS social structure [16] selected projects with a minimum of seven core developers to strike that balance. Because we have the additional constraint of needing projects that use an e-mail listserv for developers and our OSS sample sizes shrink as core developer membership grows, we were slightly more inclusive and considered a minimum team size of six core developers.

The sampling frame for this study was defined as those projects in the population of interest that were hosted in the Source Forge repository at the time of data collection and that were developed in the programming language "C". Static metrics such as those used in this paper are not well suited for cross language comparisons [8], and the choice of the "C" programming language is natural for three main reasons: First, "C" is one of the most popular programming languages. Second, code static metrics for "C" are well known and have been widely used in previous literature, and their interpretation and internal validity are generally agreed upon. Third, many tools for "C" code analysis are widely available and hence allow cross confirmation of the static code measures. Source Forge has been used as a data source in several empirical OSS studies [11-15, 52, 66]. Five hundred and eighty seven projects were included in the sampling frame. The final sample was determined by the number of projects that had active trackers for bugs and patches. The final number of projects in the sample was 29, and the data was collected in quarterly snapshots following the first modification to the SCCS, which produced 429 maximum data points. Missing data was deleted list wise for specific tests if needed.

4.2 Measurement

Source Lines of Code (SLOC) per developer, per unit of time [28] is an established metric for software development productivity. A SCCS client program was used to download the source code files for each project, obtaining snapshots of the source code base for dates corresponding to quarterly periods from the project's registration date until February 2013. Total SLOC for each project, at each of the defined points in time, were collected using a custom made script and confirmed using an off-the-shelf analyzer [80].

LOC with comments or white space were not considered in the count. The number of core developers for each instance was calculated from the SCCS log files.

The use of static code metrics can provide a way to measure code quality, which was measured by the "number of delivered bugs" per thousand lines of code [69], which is based on Halstead's "software science" metrics [41]. This metric was repeatedly empirically validated [38] and measures the expected number of pre-test defects latent in the source code. A higher defect count corresponds to lower quality.

Product popularity was determined by the number of downloads of the software for each period [86].

Effort in problem detection is measured by the number of bug reports standardized per Thousand (Kilo) Source Lines of Code (KSLOC) to compensate for bigger products naturally exhibiting more defects. Once the bug reports are in, they are assigned (or self assigned) for analysis. Most times LUs rely on fellow LUs to post questions about potential solutions before producing a patch, and it stands to reason that the more messages in the code development mailing list about a specific bug, the more effort has been spent in analyzing it (note that we control for code complexity separately). Then effort in problem analysis was measured by the number of messages in the development mailing list per bug reported. Defects are removed after a suitable patch comes in. Effort in problem removal was measured by the number of patches submitted that were obtained from the patch tracker systems' per KSLOC.

For the analysis of front loading we considered only those projects with at least two major releases (at least showing an evolution from version 0.x.x to version 2.x.x) and counted the total patches submitted until the second major release. We calculated the proportion of those patches that were submitted until the first major release (from version 0.x.x to version 1.x.x), and split the subsample by project at the median percentage. In this way, we obtained two groups; one of the projects that sent most (more than 50% of)

patches early and another which sent them in later. We ran an independent samples test between those groups with popularity, defects and productivity as dependent variables.

Team size was introduced as a control variable in the form of the number of core team members. Project tenure was also considered an additional control, measured by the number of days since the first recorded activity in the project's source code repository. The use of the type of license as control is based on previous work that found that restrictive (copyleft based) licenses were associated with increased core developer productivity in OSS [15]. The classification of OSS licenses is discussed in several resources [e.g. 82, 83]. A dummy variable was introduced with the value of 1 if the license was restrictive and zero otherwise. Code complexity can also impact new product development success [22, 88]. In the case of software, complexity is related to the intricateness of the logic design of the program flow, and this was captured by measuring average file level cyclomatic complexity [64]. All measures were log transformed to improve linearity, except for project tenure, which was inverse transformed.

5 Analysis and Results

Descriptive statistics and bivariate correlations are shown in Tables 1 and 2. The hypotheses were tested using Ordinary Least Squares (OLS) regression models. Results are shown in Table 3.

Insert Table 1, 2 and 3 here

H1a was supported. Effort in problem detection was directly associated with the number of expected defects ($p < 0.001$); i.e. inversely associated with quality. H1b and H1c were supported as well. Effort in problem detection was inversely associated with development productivity ($p < 0.05$) and directly associated with product popularity ($p < 0.001$).

H2a was supported, since effort in problem analysis was significantly ($p < 0.001$) directly associated with quality (inversely associated with number of expected defects). H2b and H2c were also supported (, p<

0.05 and $p < 0.05$ respectively); the same independent variable is inversely associated with productivity, and directly related to popularity.

H3a was supported: effort in problem removal was inversely associated with quality ($p < 0.01$). H3b was also supported, since effort in problem removal was inversely associated with productivity ($p < 0.01$). H3c was not supported, since effort in problem removal was not associated with product popularity.

As hypothesized in H4a and seen in Table 4, front loading of the problem solving stage is directly associated with quality, since those projects that solve most problems earlier in the development process have significantly fewer defects ($p < 0.01$). H4b was not supported, since there is a significant but inverse association of front loading with productivity ($p < 0.01$). A summary of all the results is in Table 5.

Insert Tables 4 and 5 here

The panel nature of the data raised the concern of error term clustering, but to test the robustness of results, they were confirmed using a non-parametric (Huber-White) regression and an AR (1) time series model. Also, results hold for teams of seven or more members.

6 Conclusions

The first goal of this study was to ascertain whether different emphases in the problem solving stages are related with product development performance outcomes, and if so, which tradeoffs are involved. The second goal was to explore whether front loading problem solving impacts performance.

Focusing on the first goal, a general finding from the results is that tapping into the LU base for any stage of problem solving always takes a toll on productivity, regardless of the problem solving stage. Although LU involvement might result in a more desirable product, processing and implementing user input is not

"free" and creates coordination unproductive inefficiencies, indicating that reducing coordination inefficiencies is a path to increase productivity.

Several avenues seem possible: for instance, in the sample used no projects had a skills matching system, and only six of them had a form of defect prioritization, but in all cases it was not consistently followed. It has been observed that in OSS the bug fixing process is highly informal and in general lacks basic controls such as an efficient bug assignment and the identification of the best match between the problem to be fixed and the developer who has the matching skills set [18], resulting in delays and confusion. To reduce the impact on productivity project leaders could implement a system with a form or system to standardize problem reports, which would make bug reports more understandable [24] and hence require less effort to be decoded into useful information. Standardizing reports would presumably reduce coordination effort by shortening the time taken to understand bugs and how/where to find them, and then reducing part of the non-value added time that would otherwise be available for new coding. Though the cited reference does not elaborate on how such a system should be set up, it mentions that it should be designed not to be a barrier to participation since in OSS, voluntary participation is critical. Bug reporting standardization should be extended to the several different channels used to report defects; for instance through the project's home page, the bug tracker, and from the application itself, which in general show different information requirements or no other instructions than to make the report as complete as possible. While maintaining several channels makes participation easier, all channels should specify what the desired information is and offer a consistent categorization of defects.

Another idea is a flagging system that classifies defects according to the skills required, as well as assigning a composite score including bug severity and priority. It has been suggested by previous work that prioritization and flagging of defects is related to the length of discussions about such problems [23], possibly due to a self-selection effect from LUs who will tend to shy away from dealing with bugs clearly

outside of their main area of expertise. Appropriately flagging defect reports would require a shorter analysis time and be more productive. When the bug report is originated LUs would assign a value for two dimensions: priority and severity, for a sort of triage that would encourage a faster resolution.

Yet another approach could be selecting which problems should be solved in house rather than opening all defect reports to user input. For some kinds of defects [5], it oftentimes is more efficient to rely on in house solutions than users'. This approach runs the risk of alienating the LU community, and care should be taken in its communication. Presumably, in house solved bugs should be those that closely match skills of the development team members or are located in code developed by them rather than in code developed by the external community. Bugs related to core functions of the software would be candidates for in house fixing. For instance, Linus Torvalds or one of his trusted "lieutenants" used to personally take care of problems in the Linux kernel in the early days of Linux's development [74]. Building up on the latter, bug solving could be segmented by having different groups of LUs treating bugs with different complexities, in a similar way as in supermarkets' service times are segmented by means of "express lanes" for customers with low average service times, which increases output.

The design of the online user community and the kinds of tools available to LUs to report and fix problems could alter this impact on productivity. Depending on the degree of two way communication between the project's management and the user base, and the kinds of user tools available to the community, different costs are associated with setting up these online communities [19]. Setting up a more coordinated community with tools that allow for bug classification, assignment, reassignment and providing multiple discussion channels demands a higher upfront investment but arguably will diminish inefficiencies once the development process starts.

Problem detection, analysis and removal are tasks that pose increasing demands on LU skills as well as the company trying to implement LU problem solving. It would be easier and cheaper to emphasize user involvement mostly in problem detection, spending less in enticing LUs for analysis and removal.

Higher problem detection effort is observed for software expected to be buggier, in line with a reactive nature of problem reporting. Presumably, not only the number of developers is important, but also the varied viewpoints they bring to the table, giving credence to Stallman's adage that "given enough eyeballs all bugs are shallow" [83], meaning that the bigger the user base, the more varied are test cases that run through the software, exposing more defects. The degree of analysis, however, decreases with the number of expected bugs.

Popularity is directly related with user participation in all stages of problem solving but not with the effort in the problem removal stage. We can also observe that there are decreasing significance levels of association between popularity and the effort spent in the three stages of the problem solving process. Popularity is more significantly associated with problem detection and less so with problem analysis and even less with problem removal. A skill-related effect cannot be ruled out, where LUs have sufficient skills to detect and report problems, but their skill set shrinks for problem analysis and removal.

User membership and hence the relative balance of skills among detection, analysis and removal could be manipulated, for instance by allowing membership by invitation [42], trying to bend the skills distribution towards more technically savvy LUs. However, attracting LUs who also have development skills is difficult, and previous research suggested the project should emphasize a social positioning in alignment with developers' motivations [13] and create opportunities that look "fun" and "challenging" to developers to enhance their inclination to participate [19]. Depending on the emphasis on problem detection or problem analysis or removal needed, the project's management could "market" the

opportunities at the different stages, by itself or for instance with the help of a community management third party, or "innovation intermediaries" [48]. Running contests to incentivize participation in specific activities has been shown as effective and could also be tried [4].

Profiling LUs can be another interesting way of making the process more effective. LU theory [32] explains "lead userness" along two dimensions: Higher expected benefits from participation and trending ahead in the use of the product. These dimensions can be measured directly or predicted from personality traits [79], and while the first dimension correlates with the likelihood of innovation, the second does with the commercial attractiveness of the innovation. Project managers could arguably measure these traits by means of a questionnaire and build LU profiles that can be useful, for instance to steer problems that need more innovative solutions toward LUs that score highly in the first dimension. Also, the positioning of LUs in those dimensions can be influenced. For instance the "higher expected benefits" dimension can be increased (as well as the LUs available), by providing information on how a product may fulfill the needs of a user in a specific environment [33].

The second goal of this study is assessing the effects of front loading on development performance. Looking at the results, front loading helps with overall expected quality but again requires a processing effort that hinders productivity. Besides being a direct empirical confirmation of the efficacy of front loading as suggested by previous work [90], this finding also has implications in project design.

For instance, attempting to front load the problem solving process would take active manipulation of the timing for bug fixing. This could be attempted by means of the product release policy. It has been observed empirically that relatively more bugs are fixed just before the planned release dates [31]. A more frequent planned release policy in the beginnings of the project could spur activity toward the front of the development process. Also, it becomes critically important to actively build a substantial user base

early in the project rather than have the user base grow passively as the product matures, which could be promoted through the early distribution of beta or demo versions.

7 Limitations and future research

One limitation of this study is that the sample is non probabilistic, which limits generalizability. However, the sample is biased toward the bigger, more active projects available in Source Forge, which better represent those projects of interest for the business community. Other limitations from the sampling design relate to the sample including only projects written in pure "C", which is a procedural language. More work is needed to confirm if the results hold for object-oriented languages such as Java.

Not supported by available data, we would have liked to measure the productivity of the problem solving process itself rather than code base growth, which could be attempted in future work. Also, the OLS models do not capture complex variable interactions, and prompt the need for structural models to be tried in future research. These results should also be replicated in the context of proprietary code development. Research on mechanisms for front loading problem solving that are actionable for project managers and their relative efficacy would be warranted, too.

8 Acknowledgements

Three anonymous reviewers deserve to be acknowledged for significantly contributing to improve this paper, with special thanks to Reviewer #1. A grant from Trinity University allowed the resources to extend and improve the database used for this paper.

9 References

[1] V. Bilgram, A. Brem, K.-I. Voigt, User centric innovations in new product development: Systematic identification of lead users harnessing interactive and collaborative online tools, International Journal of Innovation Management, 12(3) (2008) 419-458.

[2] P.M. Blau, A Formal Theory of Differentiation in Organizations, American Sociological Review, 35(2) (1970) 201-218.

[3] B.W. Boehm, Software Engineering Economics, (Prentice Hall, Englewood Cliffs, NJ, 1981).

[4] M. Bogers, J. West, Managing Distributed Innovation: Strategic Utilization of Open and User Innovation, Creativity & Innovation Management, 21(1) (2012) 61-75.

[5] V. Braun, C. Herstatt, The freedom fighters: How incumbent corporations are attempting to control user innovation, International Journal of Innovation Management, 12(3) (2008) 543-572.

[6] J.E. Carrillo, Industry Clockspeed and the Pace of New Product Development, Production and Operations Management, 14(2) (2005) 125-141.

[7] R. Chen, Member use of social networking sites — an empirical examination, Decision Support Systems, 54(3) (2013) 1219-1227.

[8] S.R. Chidamber, C.F. Kemerer, A Metrics Suite for Object-Oriented Design, IEEE Transactions on Software Engineering, 20(1994) 476-493.

[9] K.B. Clark, T. Fujimoto, Product development performance, (Harvard Business School Press, Boston, 1991).

[10] J.A. Colazo, Innovation success: An empirical study of software development projects in the context of the open source paradigm, in, (The University of Western Ontario (Canada), 2007., 2007), pp. 199 p.

[11] J.A. Colazo, Collaboration Structure and Performance in New Software Development: Findings from the Study of Open Source Projects, International Journal of Innovation Management, 14(5) (2010).

[12] J.A. Colazo, Exploring the association between temporal dispersion and virtual team performance, in: 43rd Hawaii International Conference on System Sciences, (AIS, Kauai, HI, 2010).

[13] J.A. Colazo, Y. Fang, Impact of License Choice of Open Source Software Development Activity, Journal of the American Society for Information Science and Technology, 60(5) (2009) 997-1011.

[14] J.A. Colazo, Y. Fang, Following the Sun: Temporal Dispersion and Performance in Open Source Software Project Teams, Journal of the Association for Information Systems, 11(11) (2010) 684-707.

[15] J.A. Colazo, Y. Fang, D. Neufeld, Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses, in: N.C. Romano Jr. (Ed.) Eleventh Americas Conference on Information Systems, (Omaha, NE, USA, 2005).

[16] K. Crowston, J. Howison, The Social Structure of Free and Open Source Software Development, in: International Conference on Information Systems, (2003).

[17] K. Crowston, J. Howison, The Social Structure of Free and Open Source Software Development, in: 24th International Conference on Information Systems, (Seattle, WA, 2003).

[18] K. Crowston, B. Scozzi, Bug fixing practices within free/libre open source software development teams, Journal of Database Management (JDM), 19(2) (2008) 1-30.

[19] L. Dahlander, M.G. Magnusson, Relationships between open source software companies and communities: Observations from Nordic firms, Research Policy, 34(4) (2005) 481-493.

[20] L. Dahlander, M.W. Wallin, A Man on the Inside: Unlocking Communities as Complementary Assets, Research Policy, 35(8) (2006) 1243-1259.

[21] R.C. Dailey, The Role of Team and Task Characteristics in R&D Collaborative Problem Solving and Productivity, Management Science, 24(15) (1978) 1579-1588.

[22] R.C. Dailey, The Role of Team and Task Characteristics in R&D Team Collaborative Problem Solving, Management Science, 24(15) (1978) 1579-1588.

[23] J.-M. Dalle, M. den Besten, Different bug fixing regimes? A preliminary case for superbugs, in: Open Source Development, Adoption and Innovation, (Springer, 2007), pp. 247-252.

[24] P.M. Di Gangi, M.M. Wasko, R.E. Hooker, Getting customers' ideas to work for you: Learning from Dell how to suceed with user innovation communities, MIS Quarterly Executive, 9(4) (2010) 213-228.

[25] T. Dingsøyr, S. Nerur, V. Balijepally, N.B. Moe, A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software, (2012).

[26] I. Eisenberg, Lead user research for breakthrough innovation, Research Technology Management, 54(1) (2011) 50-58.

[27] J.L. Enos, Petroleum Progress and Profits: A History of Process Innovation, (MIT Press, Cambridge, MA, 1962).

[28] N.E. Fenton, M. Neil, Software Metrics: Successes, Failures and New Directions, The journal of Systems and Software, 4(1999) 149-157.

[29] R.T. Fielding, Shared Leadership in the Apache Project, Communications of the ACM, 42(4) (1999) 42-43.

[30] K. Finstad, Analogical Problem Solving in Casual and Experienced Users: When Interface Consistency Leads to Inappropriate Transfer, Human-Computer Interaction, 23(4) (2008) 381-405.

[31] C. Francalanci, F. Merlo, Empirical analysis of the bug fixing process in open source projects, in: Open Source Development, Communities and Quality, (Springer, 2008), pp. 187-196.

[32] N. Franke, E. Von Hippel, Finding Commercially Attractive User Innovations: An Exploration of the "Lead User" Theory, in: Center for E-Business, MIT, (Cambridge, MA, 2003), pp. 31.

[33] N. Franke, E. von Hippel, M. Schreier, Finding Commercially Attractive User Innovations: A Test of Lead-User Theory, Journal of Product Innovation Management, 23(4) (2006) 301-315.

[34] S. Furst, R. Blackburn, B. Rosen, Virtual Team Effectiveness: A Proposed Research Agenda, Information Systems Journal, 9(1999) 249-269.

[35] R.B. Gallupe, A.R. Dennis, W.H. Cooper, J.S. Valacich, L.M. Bastianutti, J.F. Nunamaker, Electronic brainstorming and group size, Academy of Management Journal, 35(2) (1992) 350-369.

[36] R.L. Glass, IT Failure Rates: 70% or 10-15%?, IEEE Software, May-June 2005(2005) 112-114.

[37] M.W. Godfrey, Q. Tu, Evolution in open source software: A case study, in: The 2000 International Conference on Software Maintenance, (2000).

[38] L.L. Gremillion, Determinants of Program Repair Maintenance Requirements, Communications of the ACM, 27(8) (1984) 826-832.

[39] J.G. Guzman, A.F. del Carpió, R. Colomo-Palacios, M. Velasco de Diego, Living Labs for User-Driven Innovation, Research Technology Management, 56(3) (2013) 29-39.

[40] R.A. Guzzo, M.W. Dickson, Teams in Organizations: Recent Research on Performance and Effectiveness, Annual Review of Psychology, 47(1996) 307-338.

[41] M.H. Halstead, Elements of Software Science, (Elsevier, New York, NY, 1977).

[42] F.M. Harper, D. Frankowski, S. Drenner, Y. Ren, S. Kiesler, L. Terveen, R. Kraut, J. Riedl, Talk amongst yourselves: inviting users to participate in online conversations, in: Proceedings of the 12th international conference on Intelligent user interfaces, (ACM, Honolulu, Hawaii, USA, 2007), pp. 62-71.

[43] S.S. Hassan, Bringing Lead-User Innovations to the Market: Research and Management Implications, SAM Advanced Management Journal (07497075), 73(4) (2008) 51-58.

[44] G.J. Hidding, J. Nicholas, Reducing IT project management failures: A research proposal, in: System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on, (IEEE, 2009), pp. 1-10.

[45] C. Hienerth, C. Lettl, Exploring How Peer Communities Enable Lead User Innovations to Become Standard Equipment in the Industry: Community Pull Effects, Journal of Product Innovation Management, (s1) (2011) 175-195.

[46] M. Iansiti, Real World R&D: Jumping the Product Generation Gap, Harvard Business Review, 71(3) (1993) 131-147.

[47] L.B. Jeppesen, User Toolkits for Innovation: Consumers Support Each Other, Journal of Product Innovation Management, 22(4) (2005) 347-362.

[48] L.B. Jeppesen, K.R. Lakhani, Marginality and problem-solving effectiveness in broadcast search, Organization Science, 21(5) (2010) 1016-1033.

[49] L.B. Jeppesen, K. Laursen, The role of lead users in knowledge sharing, Research Policy, 38(10) (2009) 1582-1589.

[50] K.R. Jespersen, User involvement and open innovation: The case of decision maker openness, International Journal of Innovation Management, 14(3) (2010) 471-489.

[51] T. Johnsen, D. Ford, Managing collaborative innovation in complex networks: findings from exploratory interviews, in: 16th Annual IMP Conference, (Citeseer, 2000).

[52] S. Koch, G. Schneider, Results from Software Engineering Research into Open Source Development Projects Using Public Data, in: H.R. Hansen, W.H. Janko (Eds.) Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, (2000).

[53] S. Krishnamurthy, Cave or Community? An empirical examination of 100 mature open source projects, in: First Monday, (Bothell, WA, 2002).

[54] V. Krishnan, K.T. Ulrich, Product development decisions: A review of the literature, Management Science, 47(1) (2001) 1-21.

[55] P. Kristensson, A. Gustafsson, T. Archer, Harnessing the Creative Potential among Users, Journal of Product Innovation Management, 21(2004) 4-14.

[56] A. Kumar, A. Gupta, Evolution of developer social network and its impact on bug fixing process, in: Proceedings of the 6th India Software Engineering Conference, (ACM, 2013), pp. 63-72.

[57] K. Lakhani, E. von Hippel, How Open Source software works: "Free" user-to-user assistance, Research Policy, In Press(2002).

[58] N. Levina, Collaborating on Multiparty Information Systems Development Projects: A Collective Reflection-in-Action View, Information Systems Research, 16(2) (2005) 109-130.

[59] G.L. Lilien, P.D. Morrison, K. Searls, M. Sonnack, E. Von Hippel, Performance Assessment of the Lead User Idea-Generation Process for New Product Development, Management Science, 48(8) (2002) 1042-1059.

[60] C. Lüthje, C. Herstatt, The Lead User Method: An Outline of Empirical Findings and Issues for Future Research, R&D Management, 34(5) (2004) 553-568.

[61] D. Mahr, A. Lievens, Virtual lead user communities: Drivers of knowledge creation for innovation, Research Policy, 41(1) (2012) 167-177.

[62] V. Mandić, M. Oivo, P. Rodríguez, P. Kuvaja, H. Kaikkonen, B. Turhan, What Is Flowing in Lean Software Development?, in: Lean Enterprise Software and Systems, (Springer, 2010), pp. 72-84.

[63] G. Marchi, C. Giachetti, P. de Gennaro, Extending lead-user theory to online brand communities: The case of the community Ducati, Technovation, 31(8) (2011) 350-361.

[64] T. McCabe, A Software Complexity Measure, IEEE Transactions on Software Engineering, SE-2(4) (1976) 308-320.

[65] E.F. McDonough, G. Barczak, The Effects of Cognitive Problem-Solving Orientation and Technological Familiarity on Faster New Product Development, Journal of Product Innovation Management, 9(1992) 44-52.

[66] A. Mockus, R.T. Fielding, J. Herbsleb, Two case studies of Open Source Software development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, 11(3) (2002) 309-346.

[67] J.M. Morgan, J.K. Liker, The Toyota product development system, (Productivity press New York, 2006).

[68] G.J. Myers, C. Sandler, T. Badgett, The art of software testing, (John Wiley & Sons, 2011).

[69] L. Ottenstein, Predicting Numbers of Errors Using Software Science, ACM SIGMETRICS Performance Evaluation Review, 10(1) (1981) 157-167.

[70] S. Pai Cheng, H. Hsin-Yun, F. Cheng-Kiang, Lead user participation in brand community: The case of Microsoft MVPS, International Journal of Electronic Business Management, 8(4) (2010) 323-331.

[71] M.K. Poetz, M. Schreier, The value of crowdsourcing: can users really compete with professionals in generating new product ideas?, Journal of Product Innovation Management, 29(2) (2012) 245-256.

[72] R. Purushothaman, D.E. Perry, Understanding the Software Development Process by Analysis of Changed Lines, in, (University of Texas at Austin, 2002).

[73] E.S. Raymond, The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, 2 ed., (O'Reilly, Sebastapol, CA, 2001).

[74] E.S. Raymond, W.C. Trader, Linux and Open-Source Success, IEEE Software, 16(1) (1999) 85-89.

[75] W. Scacchi, Computer game mods, modders, modding, and the mod scene, First Monday, 15(5) (2010).

[76] M. Schreier, C. Fuchs, D.W. Dahl, The Innovation Effect of User Design: Exploring Consumers' Innovation Perceptions of Firms Selling Products Designed by Users, Journal of Marketing, 76(5) (2012) 18-32.

[77] M. Schreier, S. Oberhauser, R. Prügl, Lead users and the adoption and diffusion of new products: Insights from two extreme sports communities, Marketing Letters, 18(1/2) (2007) 15-30.

[78] M. Schreier, R. Prügl, Extending Lead-User Theory: Antecedents and Consequences of Consumers' Lead Userness, Journal of Product Innovation Management, 25(4) (2008) 331-346.

[79] M.C. Schuhmacher, S. Kuester, Identification of Lead User Characteristics Driving the Quality of Service Innovation Ideas, Creativity & Innovation Management, 21(4) (2012) 427-442.

[80] Scitools, Understand, in, (Scientific Toolworks Inc., 2012), pp. Static Metrics Analyzer.

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

[82] A.M. St. Laurent, Understanding Open Source and Free Software Licensing, (O'Reilly, Sebastopol, CA, 2004).

[83] R.M. Stallman, L. Lessig, Free software, free society: Selected essays of Richard M. Stallman, (GNU Press, Boston, MA, 2002).

[84] G. Stepanek, Programming secrets: Why software projects fail, (Apress, 2011).

[85] D.W. Stewart, P.N. Shamdasani, Focus Groups: Theory and Practice, (Sage Publications, Newbury Park, CA, 1990).

[86] K.J. Stewart, OSS Success: From Internal Dynamics to External Impact, in: 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering, (Edinburgh, UK, 2004).

[87] M.R. Subramani, B. Rajagopalan, Knowledge-sharing and influence in online social networks via viral marketing, Communications of the ACM, 46(12) (2003) 300-307.

[88] M. Swink, Threats to New Product Manufacturability and the Effects of Development Team Integration Processes, Journal of Operations Management, 17(1999) 691-709.

[89] S. Thomke, Managing Experimentation in the Design of New Products, Management Science, 44(6) (1998) 743-762.

[90] S. Thomke, T. Fujimoto, The Effect of "Front-Loading" Problem-Solving on Product Development Performance, Journal of Product Innovation Management, 17(2000) 128-142.

[91] S.H. Thomke, The Role of Flexibility in the Design of New Products: An Empirical Study, (Division of Research, Harvard Business School, 1996).

[92] G.L. Urban, E. Von Hippel, Lead User Analyses for the Development of New Industrial Products, Management Science, 34(5) (1988) 569-582.

[93] E. Von Hippel, The Dominant Role of Users in the Scientific Instrument Innovation Process, Research Policy, 5(1976) 212-239.

[94] E. Von Hippel, Lead Users: A Source of Novel Product Concepts, Management Science, 32(7) (1986) 791-805.

[95] E. Von Hippel, Innovation by user Communities: Learning from Open Source Software, MIT Sloan Management Review, 42(2001) 82-86.

[96] E. Von Hippel, Democratizing Innovation, (MIT Press, Cambridge, MA, 2005).

[97] Y. Wang, D. Li, Testing the moderating effects of toolkits and user communities in personalization: The case of social networking service, Decision Support Systems, 55(1) (2013) 31-42.

[98] K.D. Welker, P.W. Oman, Software Maintainability Metrics Models in Practice, Journal of Defense Software Engineering, 8(11) (1995) 19-23.

[99] A. Westerski, T. Dalamagas, C.A. Iglesias, Classifying and comparing community innovation in Idea Management Systems, Decision Support Systems, 54(3) (2013) 1316-1326.

[100] S.E. White, J.E. Dittrich, J.R. Lang, The effects of group decision-making process and problem situation complexity on implementation attempts, Administrative Science Quarterly, (1980) 428-440.

[101] L. Zhao, S. Elbaum, Quality assurance under the open source development model, Journal of Systems and Software, 66(1) (2003) 65-75.

[102] Y.A. Zhao, H. Liu, "Not My Bug!" in Open Source World.

Table 1: Descriptives

Construct	Units	Min	Mean	Max	SD
Popularity	Downloads	0	10,212	1,380,846	77,908
Productivity	LOC per dev	0	648	112,841	80
Quality	Bugs/KSLOC	0	46	122	1.5
Problem detection effort	Bug reports per KSLOC	0	1.03	3.5	0.7
Problem analysis effort	Messages per bug report	0	6.25	35	1.1
Problem removal effort	Patches per KSLOC	0	1.01	5.75	1.2
Complexity	McCabe's complexity	1	13.29	169	17
Team size	# of developers	6	9.44	13	2.2
Project tenure	Months	3	27.87	135	6.6

Table 2: Bivariate Correlations

Downloads	1		.350	***	.488	***	.307	***	.361	***	.189	***	.145	***	.292	***	.118	***	.332	***
LOC per dev	.350	***	1		.363	***	.141	***	.152	**	.063		.089	**	.304	***	.008		.124	***
B/KSLOC	.488	***	.363	***	1		.219	***	.155	***	.246	***	.221	***	.273	***	.070	**	.241	***
Bug reports per KSLOC	.307	***	.141	***	.219	***	1		-.296	***	.431	***	-.038		.262	***	-.070	*	.017	
Messages per bug report	.361	***	.152	**	.155	***	-.296	***	1		-.059		.058		.305	***	.138	***	.127	**
Patches per KSLOC	.189	***	.063		.246	***	.431	***	-.059		1		-.051		.104	**	.068	*	.003	
Complexity	.145	***	.089	**	.221	***	-.038		.058		-.051		1		-.020		-.066	*	-.021	
Core developers	.292	***	.304	***	.273	***	.262	***	.305	***	.104	**	-.020		1		-.062	*	-.142	***
License	.118	***	.008		.070	**	-.070	*	.138	***	.068	*	-.066	*	-.062	*	1		.027	
Tenure	.332	***	.124	***	.241	***	.017		.127	**	.003		-.021		-.142	***	.027		1	
Mean	2.457		2.812		1.662		.014		.796		.005		.975		.423		.815		-.221	
SD	1.462		1.133		.504		.045		.773		.020		.317		.270		.389		.247	
***	$p < 0.001$																	
**	$p< 0.01$																	
*	$p < 0.05$																	

Table 3: Regressions

	Popularity		Productivity		Expected defects	
Intercept	2.977	***	2.085	***	1.758	***
Log, bug reports per KSLOC	5.960	***	-.767	*	.658	***
Log, messages per bug report	.213	*	-.021	*	-.045	**
Log, patches per KSLOC	-.841		-.821	**	1.294	**
Log, cyclomatic complexity	-.049		.031		.045	*
Log, core developers	.379	*	1.440	***	.006	
License type	.192	*	.111	*	.050	*
Tenure	1.695	***	.133		-.008	
N	429		381		429	
F	25.1	***	7.8	***	11.4	***
R^2	.300		.120		.160	
***	p < .001					
**	p< 0.01					
*	p < 0.05					

Table 4: T-tests

	Means		t	
	Early	Late		
Productivity	2.34	2.74	-6.8	***
Defects per KSLOC	0.87	2.66	-12.4	***

N = 215
All variables transformed
Equal variances not assumed

Table 5: Tests Summary

	Association with:		
	Quality	Productivity	Popularity
Problem detection	inverse	inverse	direct
Problem analysis	direct	inverse	direct
Problem removal	inverse	inverse	Not sig.
Front Loading	direct	inverse	Not tested